НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою
«Інженерія програмного забезпечення
комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Модифікований метод та програмне забезпечення для дискретного логарифмування»

Виконала:
студентка IV курсу, групи КП-62
Мірошник Віталіна Ігорівна _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри СПіСКС, к.н.т., доцент,
Клятченко Ярослав Михайлович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____ 2019 р.

ЗАВДАННЯ

на дипломну роботу студенту

Мірошник Віталіні Ігорівні

1. Тема роботи «Модифікований метод та програмне забезпечення для дискретного логарифмування», керівник роботи Онай Микола Володимирович, доцент кафедри ПЗКС, к.т.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1181-с

2. Термін подання студентом роботи «11» червня 2020 р.

3. Вихідні дані до роботи:
   - алгоритми реалізації класичних методів дискретного логарифмування.

4. Зміст роботи:
   - аналіз існуючих рішень;
   - розроблення модифікованого методу дискретного логарифмування;
   - обґрунтування засобів реалізації;
   - розроблення програмного забезпечення для реалізації та аналізу продуктивності роботи запропонованого модифікованого методу дискретного логарифмування.

5. Перелік обов'язкового ілюстративного матеріалу:
   - принцип роботи асиметричної криптосистеми;
   - діаграма класів розроблюваного застосунку;
   - графіки, що ілюструють аналіз ефективності роботи методів дискретного логарифмування;
   - графік, що ілюструє порівняння модифікованого методу і класичного.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М. В., доцент кафедри ПЗКС, к.т.н., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за тематикою роботи та збір даних	16.11.2019	
2.	Проведення порівняльного аналізу математичних методів дискретного логарифмування	09.12.2019	
3.	Підготовка матеріалів першого розділу дипломної роботи	30.12.2019	
4.	Розробка модифікації методу дискретного логарифмування	16.01.2020	
5.	Підготовка матеріалів другого розділу дипломної роботи	10.02.2020	
6.	Підготовка тез доповіді на конференцію	20.02.2020	
7.	Підготовка матеріалів третього розділу дипломної роботи	10.03.2020	
8.	Підготовка матеріалів четвертого розділу дипломної роботи	11.04.2020	
9.	Підготовка графічної частини дипломної роботи	19.05.2020	
10.	Оформлення дипломної роботи	26.05.2020	

Студент Віталіна МІРОШНИК

Керівник роботи Микола ОНАЙ

АНОТАЦІЯ

Дана дипломна робота присвячена дослідженню існуючих методів та розробленню модифікованого методу дискретного логарифмування.

У роботі проведено аналіз роботи асиметричних криптосистем, описано принцип роботи з відкритим та закритим ключами та їх залежність один від одного. Проаналізовано та програмно реалізовано детерміновані методи дискретного логарифмування: метод підстановки, проста формула, алгоритм узгодження, алгоритм Поліга-Геллмана, ϱ-метод Поларда, а також оцінена їх складність. Кожен із них по-різному підходить для вирішення задачі: метод перебору і проста формула можуть бути використані, коли час роботи неважливий, алгоритм узгодження і алгоритм Поліґа-Геллмана – для математичного застосування з невеликими числами, а ϱ-метод Поларда – у задачах, пов’язаних з наборами великих чисел, наприклад, у криптографії. Запропоновано модифікований метод дискретного логарифмування, який відрізняється від існуючого, по-перше, використанням модулярної арифметики замість арифметики з плаваючою крапкою, і, по-друге, відсутністю необхідності у піднесенні до степеня на останньому кроці.

У даній дипломній роботі розроблено десктоп застосунок для проведення експериментальних досліджень, який реалізує існуючі методи дискретного логарифмування та запропоновану модифікацію одного з методів. Даний застосунок дозволяє виконувати аналіз швидкодії реалізованих методів для різних наборів даних, зокрема чисел, що мають різну бітову довжину.

ABSTRACT

This graduate work is devoted to the study of methods of existing methods and the development of a modified method of discrete logarithm.

The graduate work provides an analyze of the operation of asymmetric cryptosystems, describes the principle of working with public and private keys and their dependence on each other. Deterministic methods of discrete logarithm such as brute-force search, simple formula, coordination algorithm, Pohlig-Hellman algorithm, Pollard`s ϱ-algorithm, are analyzed, programmatically implemented and their complexity is estimated. Each of them is differently suited to solve the problem: the brute-force search and a simple formula can be used when the operating time is irrelevant, the matching algorithm and the Pohlig-Hellman algorithm – for mathematical applications with small numbers, and the Pollard`s ϱ-algorithm – in problems associated with sets of large numbers, for example, in cryptography. A modified method of discrete logarithm is proposed, which differs from the existing one, firstly, by using modular arithmetic instead of floating-point arithmetic, and secondly, by the absence of the need to elevate to the power in the last step.

In this graduate work a desktop application that implements existing methods of discrete logarithm, as well as a proposed modification of one of the methods, which allows for experimental research, in particular to compare their performance on different numbers of digits, is developed and researched.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Криптосистема – система шифрування, що виконує двонаправлені криптоперетворення даних і має функції посилення захищеності даних та полегшення роботи з криптоалгоритмами для людини.

Детермінований алгоритм – алгоритм, який при однакових вхідних даних буде видавати відповідно однакові вихідні дані.

Логарифм числа – це степінь, до якого необхідно піднести основу a, щоб отримати число b. Позначається $\log_a b$.

Mod операція – це знаходження остачі після ділення одного цілого числа на інше.

Дискретний логарифм – це розв'язок рівняння $a^x \equiv b(\mathrm{mod}\, p)$ скінченної циклічної групи G.

Конгруентність чисел – це їх рівність за модулем. Позначається $\equiv$.

Оцінка складності алгоритму $O(f(n))$ – нотація для позначення часу, необхідного для виконання алгоритму і є підрахунком елементарних операцій з опущенням констант.

НСД – найбільший спільний дільник невід'ємних чисел.

Псевдовипадкові числа – числа, які мають схожі властивості до випадкових, але насправді у шляху до їх отримання лежить зовсім невипадковий алгоритм.

Об'єктно-орієнтоване програмування (ООП) – це методологія програмування, що базується на концепції об'єктів та взаємозв'язків між ними. Об'єкти зазвичай включають у себе атрибути (поля, що містять дані) та функції (код у вигляді процедур).

Високорівнева мова програмування у комп'ютерних науках – це мова програмування з високим рівнем абстракції, що значно полегшує роботу програміста.

IDE – це програмний застосунок, який включає в себе поєднання текстового редактору та комплексних можливостей для програмування, таких як редактор коду, інструменти автоматизованої побудови проекту та допомога у зневадженні.

Singleton – шаблон проєктування, що гарантує створення лише одного екземпляру класу.

Імплементація – програмна реалізація.

ВСТУП

Криптографія – це наука про шифрування та захищену передачу інформації. Вона бере свій початок ще в найдавніші часи, коли людство почало використовувати прості шифри для відправки таких повідомлень, які не мали б бути перехоплені. Такі шифри часто використовувались у війнах для відправки наказів та даних розвідки, а також для спілкування шпигунів та дипломатів. Ворог міг бачити шифри, але тільки у «правильних» людей були ключі для дешифрування повідомлень.

Криптографія еволюціонувала від найпростіших шифрів, у яких просто мінялись місцями літери або навіть абетки, або використовувався шифр Цезаря, який шифрував повідомлення шляхом заміни літери на зсунуту на кілька позицій алфавіту, до більш складних – шифрувальний диск Альберті, поліалфавітні шифри, циліндр Джефферсона, а у роки Другої світової війни – роторні машини, серед яких найвідомішим розробленням стала німецька – Енігма.

З початком комп’ютерної ери більшість минулих шифрів стала непотрібною, адже базувалась в першу чергу на письмових текстах, на зміну яким прийшли нулі і одиниці – тепер мова йде не про літери, а про дані, представлені у двійковому вигляді, що дуже сильно підвищило складність шифрів.

Тільки в 70-х роках минулого століття почались серйозні дослідження в галузі криптографії, яка з тих пір посіла чільне місце як засіб захисту даних та передачі інформації в комп’ютерних мережах через використання асиметричних систем, про які йтиметься далі. Вони використовують один ключ для шифрування і непов’язаний з ним другий для дешифрування.

У зв'язку з цим виникає необхідність розширити дослідження в галузі криптографії шляхом аналізу існуючих методів дискретного логарифмування, що лежать в основі асиметричних криптосистем, а також

модифікація методу для зменшення затрат ресурсів на розв’язання даної задачі.

У цій дипломній роботі буде досліджено існуючі методи дискретного логарифмування та їх модифікації з визначенням їх ефективності, запропоновано модифікований метод, що має менший час роботи порівняно з існуючими.

МЕТА ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

Метою дипломної роботи є розроблення модифікованого методу дискретного логарифмування та програмна реалізація існуючих і модифікованого методів для аналізу швидкості їх роботи.

Науково-практична задача, що розв’язується у даній дипломній роботі, має наступні завдання:

- аналіз існуючих методів дискретного логарифмування;
- модифікація одного з методів дискретного логарифмування для зменшення часу роботи;
- розроблення застосунку для реалізації методів і їх модифікацій;
- перевірка правильності методів та їх модифікацій шляхом аналітичного та програмного тестування;
- проведення експериментальних досліджень методів дискретного логарифмування;
- аналіз успішності модифікації шляхом тестування на різних наборах даних.

1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ

1.1. Загальний аналіз проблеми

Асиметричні криптосистеми – це системи криптографічного захисту інформації та даних, які базуються на тому, що для шифрування даних використовується один ключ, а для дешифрування – зовсім інший. Такі системи також називають криптосистемами з відкритим ключем. Вони дозволяють передавати захищені дані без необхідності обмінюватись спеціальним секретним ключем між сторонами. Для дешифрування повідомлення чи даних необхідно знати тільки таємний ключ, який не використовувався для шифрування.

У асиметричних криптосистемах використовується підхід, що використовує два ключі – відкритий та закритий. Перший ключ має за мету зашифрувати дані і є відкритим, тобто він є видимим усім користувачам даної системи, яка шифрує дані. Але дешифрувати дані за допомогою цього ключа не представляється можливим. Задачею дешифрування даних займається користувач з другим ключем – закритим, який є недоступним для всіх. З відкритого ключа майже неможливо отримати закритий і навпаки. На рис. 1 наведено принцип роботи відкритого та закритого ключів.

Рис. 1. Принцип роботи відкритого та закритого ключів

Принцип роботи асиметричної криптосистеми (рис. 2):

1. Користувач 1 генерує два ключі – відкритий та закритий.
2. Користувач 1 передає відкритий ключ публічним каналом користувачу 2.
3. Користувач 2 шифрує своє повідомлення відкритим ключем, переданим йому користувачем 1 у криптотекст.
4. Користувач 2 надсилає криптотекст публічним каналом користувачеві 1.
5. Користувач 1 дешифрує криптотекст своїм секретним закритим ключем.

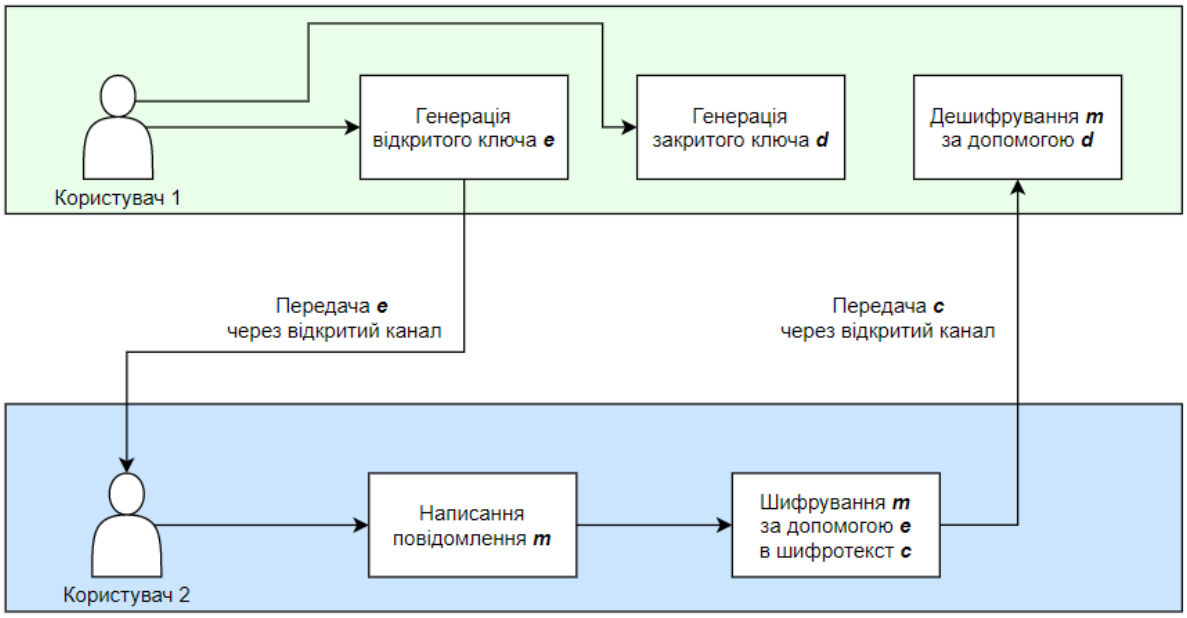


Рис. 2. Принцип роботи асиметричної криптосистеми

Завдяки використанню mod з великими числами можна створити односторонні функції. Це такий вид функцій, який дозволяє легко обчислювати значення тільки в одному напрямку. В зворотньому напрямку ця операція або неможлива, або займає значно більше часу.

Головна проблема – вразливість криптосистем з відкритим ключем. Перша причина – теоретично доведено, що дискретний логарифм можна обчислити за поліноміальний час. Якщо поліноміальний алгоритм

дискретного логарифмування буде реалізовано на практиці, це ставить під загрозу придатність асиметричних криптосистем для довгострокового захисту інформації. Другою причиною є відсутність універсального підходу до вирішення логарифмічних рівнянь, що в свою чергу спричинено недостатньою кількістю досліджень працездатності, недостатньою кількістю програмних реалізацій існуючих алгоритмів і відсутністю достатньої кількості модифікацій цих алгоритмів.

1.2. Аналіз існуючих рішень для даної задачі

Дискретний логарифм – теоретико-груповий аналог звичайного логарифму. Зокрема, звичайний логарифм $\log_a(b)$ – це розв'язок рівняння

$$a^x = b$$

у полі дійсних або комплексних чисел. Аналогічно, якщо g і h елементи зі скінченної циклічної групи G, тоді розв'язок x рівняння

$$g^x = h$$

зветься дискретним логарифмом h за основою g в групі G.

1.2.1. Метод перебору

Розглянемо задачу дискретного логарифмування в кільці класів рівності за модулем простого числа. Нехай дано порівняння

$$2^x \equiv 7 \pmod{19}. \qquad (1.1)$$

Розв’яжемо задачу методом перебору [1]. Випишемо таблицю всіх степенів числа 2 і їх остачі від ділення на 19 (табл. 1).

Таблиця 1

Степені числа 2 і їх остачі від ділення на 19

$2^1 \equiv 2 \pmod{19}$
$2^2 \equiv 4 \pmod{19}$
$2^3 \equiv 8 \pmod{19}$
$2^4 \equiv 16 \pmod{19}$
$2^5 \equiv 32 \equiv 13 \pmod{19}$
$2^6 \equiv 64 \equiv 7 \pmod{19}$
$2^7 \equiv 128 \equiv 14 \pmod{19}$

Розв'язком порівняння (1.1) є $x = 6$, оскільки $2^6 \equiv 7 \pmod{19}$.

1.2.2. Проста формула

Також розв’язок можна знайти по формулі [2]

$$\log_a b \equiv \sum_{j=1}^{p-2} \frac{b^j}{1-a^j} (\mathrm{mod}\ p-1). \qquad (1.2)$$

Спробуємо вирішити порівняння (1.1) за формулою (1.2):

$$x \equiv \log_2 7 (\mathrm{mod} 19) \equiv \sum_{j=1}^{17} \frac{7^j}{1-2^j} (\mathrm{mod} 18) \equiv 6.$$

На практиці задача часто полягає в роботі зі значно більшими числами. Методом перебору можна вирішити задачу за $O(p)$ арифметичних операцій, а вищезгаданою формулою – ще повільніше. Для модуля, який є великим числом, вирішення задачі є надто повільним і для розв’язання задачі необхідно шукати оптимальніші по затратам часу рішення.

Розглянемо інші детерміновані методи розв’язання логарифмічного рівняння.

1.2.3. Алгоритм узгодження [1]

1. Порахуємо

$$H = \lfloor p^{1/2} \rfloor + 1.$$

2. Знайдемо

$$c \equiv a^H \pmod{p}.$$

3. Складемо та впорядкуємо таблицю

$$c^u \pmod{p},\ 1 \le u \le H.$$

4. Складемо та впорядкуємо таблицю

$$b \cdot a^v \pmod{p},\ 0 \le v \le H.$$

5. Шукаємо співпавші елементи в двох таблицях, для них

$$c^u \equiv b \cdot a^v \pmod{p},$$

звідки

$$a^{Hu-v} \equiv b \pmod{p}.$$

6. Розв'язком є

$$x \equiv Hu - v \pmod{p}.$$

Нехай дано порівняння

$$3^x \equiv 13 \pmod{23}. \qquad (1.3)$$

Спробуємо вирішити його за допомогою алгоритму узгодження:

1. $H = \lfloor 23^{1/2} \rfloor + 1 = 5.$

2. $c \equiv 3^5 \pmod{23} = 13.$

3. Складемо та впорядкуємо таблицю $13^u \pmod{23},\ 1 \le u \le 5$ (табл. 2).

Таблиця 2

Формування таблиці передобчислень для алгоритму узгодження

u	$\boldsymbol{c^u \pmod{p}}$
5	4

Продовження табл. 2

2	8
3	12
1	13
4	18

4. Складемо та впорядкуємо таблицю $13 \cdot 3^v \pmod{23},\ 0 \le v \le 5$ (табл. 3).

Таблиця 3

Формування таблиці передобчислень для алгоритму узгодження

v	$b \cdot a^v \pmod{p}$
2	2
3	6
5	8
0	13
1	16
4	18

5. Перший співпавший елемент у двох таблицях (табл. 2, табл. 3) – 8 при $u = 2$ і $v = 5$, звідки $3^{5\times 2-5} \equiv 13 \pmod{23}$.

6. Розв'язком порівняння (1.3) є $x \equiv 5 \times 2 - 5 \pmod{23} \equiv 5.$

Наведеним вище алгоритмом можна вирішити задачу за

$$O(p^{1/2} \log p)$$

арифметичних операцій.

1.2.4. Алгоритм Поліґа-Геллмана [3]

1. Розкладемо $p-1$ на прості множники:

$$p-1=\prod_{i=1}^{s} q_i^{\alpha_i}.$$

2. Складемо таблицю чисел $r_{q,i}$ для кожного простого числа q такого, що $q \mid p-1$:

$$r_{q,j} = a^{j(p-1)/q} (\mathrm{mod}\ p),\ j \in [0, q-1].$$

3. Для кожного простого q знаходимо $\log_a b (\mathrm{mod}\, q^{\alpha})$:

$$x \equiv \log_a b (\mathrm{mod}\, q^{\alpha}) \equiv x_0 + x_1 q + \ldots + x_{\alpha-1} q^{\alpha-1} (\mathrm{mod}\, q^{\alpha}),$$

де $0 \le x_i \le q-1$. Тоді з наступного виразу знаходимо x_0,

використовуючи таблицю з п. 1:

$$b^{(p-1)/q} \equiv a^{x_0 (p-1)/q} (\mathrm{mod}\ p).$$

Далі за вищезгаданою таблицею знаходимо x_1, … x_i з виразу:

$$(ba^{-x_0 - x_1 q - \ldots - x_{i-1} q^{i-1}})^{(p-1)/q^{i+1}} \equiv a^{x_i (p-1)/q} (\mathrm{mod}\ p).$$

4. Знайшовши $\log_a b (\mathrm{mod}\, q_i^{\alpha_i})$ для $i \in [1, s]$, знаходимо

$$\log_a b (\mathrm{mod}\ p-1)$$

за китайською теоремою про остачі.

Для прикладу вирішимо порівняння (1.1) за алгоритмом Поліґа-Геллмана.

1. Розкладемо $p-1$ на прості множники: $18 = 2 \times 3^2$.

2. Складемо таблицю чисел $r_{q,i}$ для кожного простого числа q $r_{q,j} = 2^{18 \times j/q} (\mathrm{mod} 19),\ j \in [0, q-1]$ (табл. 4).

Таблиця 4

Формування таблиці передобчислень для алгоритму Поліґа-Геллмана

q	j	$r_{q,j}$
2	0	1
	1	18
3	0	1
	1	7
	2	11

3. Для кожного простого q знаходимо $\log_2 7 (\mathrm{mod}\, q^{\alpha})$ (табл. 5).

Таблиця 5

Формування таблиці передобчислень для алгоритму Поліґа-Геллмана

q	x_i	x
2	0	0
	0	
3	0	6
	2	

4. За китайською теоремою про остачі знаходимо розв’язок системи:

$$\begin{cases} x \equiv 0 (\mathrm{mod}\, 2); \\ x \equiv 6 (\mathrm{mod}\, 9); \end{cases}$$

$$x \equiv 6 (\mathrm{mod} 18) \equiv 6 (\mathrm{mod} 19).$$

Алгоритмом Поліґа-Геллмана можна вирішити задачу за

$$O(\sum_{i=1}^{s} \alpha_i (\log p + q_i))$$

арифметичних операцій.

1.2.5. ϱ-метод Поларда для дискретного логарифмування [4]

1. Розглянемо три числові послідовності $\{u_i\}$, $\{v_i\}$, $\{z_i\}$, $i = 1, 2, \ldots$, визначені наступним чином:

$$u_0 = v_0 = 0, \quad z_0 = 1;$$

$$u_{i+1} \equiv \begin{cases} u_i + 1 (\mathrm{mod}\ p-1), & 0 < z_i \le p/3; \\ 2u_i (\mathrm{mod}\ p-1), & p/3 < z_i \le \frac{2}{3} p; \\ u_i (\mathrm{mod}\ p-1), & \frac{2}{3} p < z_i \le p; \end{cases}$$

$$v_{i+1} \equiv \begin{cases} v_i (\mathrm{mod}\ p-1), & 0 < z_i \le p/3; \\ 2v_i (\mathrm{mod}\ p-1), & p/3 < z_i \le \frac{2}{3} p; \\ v_i + 1 (\mathrm{mod}\ p-1), & \frac{2}{3} p < z_i \le p; \end{cases}$$

$$z_{i+1} \equiv b^{u_{i+1}} a^{v_{i+1}} (\mathrm{mod}\ p-1).$$

2. Розглядаємо набори $(z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$, $i = 1, 2, \ldots$, і шукаємо таке значення i, для якого $z_i = z_{2i}$, звідки

$$a^m \equiv b^n (\mathrm{mod}\ p), \tag{1.4}$$

де $m = u_i - u_{2i}$, $n = v_{2i} - v_i$.

3. Знайдемо $d = \mathrm{НСД}(m, p-1) = \lambda \times m + \mu \times (p-1)$ за допомогою розширеного алгоритму Евкліда.

4. Піднесення (1.4) до степеню λ дасть $a^d \equiv b^{\lambda n} (\mathrm{mod}\ p)$, де λn є формою dx. Звідси

$$b \equiv a^k \theta^w (\mathrm{mod}\ p),$$

де $\theta \equiv a^{(p-1)/d}$, $w \in [0; d]$.

5. Знаходимо w методом перебору, тоді

$$x \equiv \frac{\lambda \times n + w \times (p-1)}{d} \equiv \frac{k + w \times (p-1)}{d}.$$

ϱ-методом Поларда можна вирішити задачу за $O(p^{1/2})$ арифметичних операцій.

1.2.6. Результати аналізу

Поліноміального алгоритму для розв'язання цієї задачі поки не існує. Виходячи з отриманих даних після проведення аналізу, всі методи є різними і мають різну складність і швидкодію. Кожен із них по-різному підходить для вирішення задачі: метод перебору і проста формула можуть бути використані, коли час роботи неважливий, алгоритм узгодження і алгоритм Поліґа-Геллмана – для калькулятора, що реалізує модульну арифметику, а ϱ-метод Поларда – у задачах, пов’язаних з наборами довгих чисел, наприклад, у криптографії. Тож основною задачею є програмна реалізація вищенаведених методів для пошуку модифікації одного з алгоритмів і створення модифікації, що буде якнайкраще підходити до сучасних реалій і можливостей криптоаналізу.

1.3. Висновки

У першій частині цього розділу наведено загальний аналіз роботи асиметричних криптосистем та їх призначення. Проаналізовано принцип роботи з відкритим та закритим ключами та їх залежність один від одного. Також обґрунтовано проблеми, пов'язані з використанням криптосистем з відкритим ключем.

У другій частині поточного розділу наведено існуючі методи дискретного логарифмування: метод підстановки, проста формула, алгоритм узгодження, алгоритм Поліґа-Геллмана, ϱ-метод Поларда для дискретного логарифмування, а також обґрунтовано, що вони мають різну складність і швидкодію і можуть бути застосованими у різних сферах.

Для реалізації завдань даної дипломної роботи потрібно:

- модифікувати ϱ-метод Поларда для дискретного логарифмування;

- розробити застосунок, який знаходитиме корінь логарифмічного рівняння різними методами;
- реалізувати графічно-користувацький інтерфейс;
- дослідити існуючі методи дискретного логарифмування аналітичним шляхом;
- дослідити обчислювальну складність існуючих методів дискретного логарифмування шляхом їх програмної реалізації;
- загальне тестування системи на правильність даних;
- протестувати алгоритми дискретного логарифмування на різних наборах даних;
- протестувати алгоритми дискретного логарифмування з замірами часу.

2. РОЗРОБЛЕННЯ МОДИФІКОВАНОГО МЕТОДУ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

2.1. Аналіз класичного ϱ-методу Поларда для дискретного логарифмування

ϱ-метод Поларда для дискретного логарифмування вважається ефективним для вирішення задачі дискретного логарифмування. Він був опублікований у 1978 році британським математиком Джоном Полардом у науковій статті під назвою «Monte Carlo Methods for Index Computation (mod p)» наукового журналу «Mathematics of Computation»[2].

ϱ-метод Поларда базується на визначенні числової послідовності, яка зациклюється при певному *i*. Названо метод саме так, тому що на схематичному зображенні точки послідовності візуально утворюють грецьку букву ϱ (рис. 3).

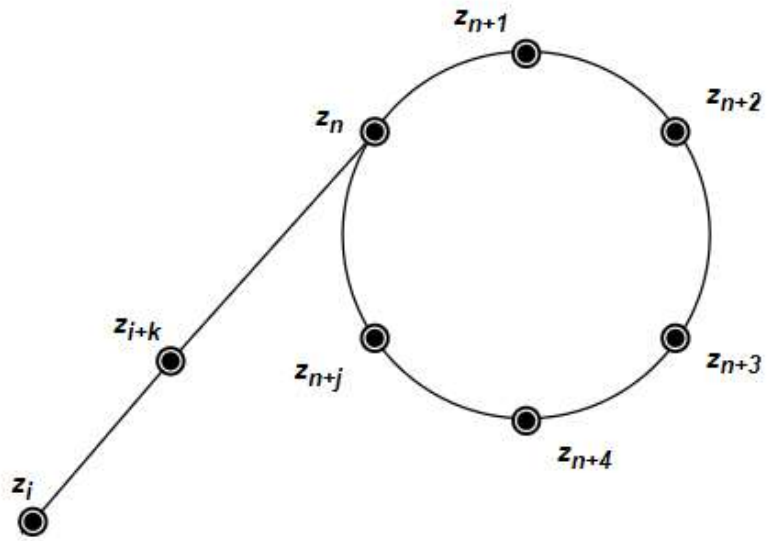


Рис. 3. Схематичне зображення точок послідовності ϱ-методу Поларда

У пункті 1.2.5 наведено ϱ-метод Поларда для дискретного логарифмування. Суть цього методу полягає у побудові псевдовипадкових послідовностей з подальшим пошуком співпавших елементів для

розрахунку шуканого дискретного логарифму, аналогічно до ϱ-методу Поларда для факторизації цілих чисел.

Розглянемо приклад вирішення порівняння (1.1) методом, запропонованим у оригінальній статті.

1. Розглянемо набори $(z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$, сформовані з трьох числових послідовностей $\{u_i\}$, $\{v_i\}$, $\{z_i\}$ (табл. 6).

Таблиця 6

Формування таблиці передобчислень псевдовипадкових множин для ϱ-методу Поларда

i	u_i	v_i	z_i
0	0	0	1
1	1	0	7
2	2	0	11
3	4	0	7
4	8	0	11
5	16	0	7

2. $z_i = z_{2i} = 11$ досягається при $i = 2$. $m = u_2 - u_4 = -6 (\mathrm{mod} 18) = 12$, $n = v_4 - v_2 = 0 (\mathrm{mod} 18) = 0$.

3. За допомогою розширеного алгоритму Евкліда знайдемо

$$d = \text{НСД}(12, 19-1) = -1 \times 12 + 1 \times (19-1) = 6.$$

4. $7 \equiv 2^{-1 \times 0} 2^{w \times (19-1)/6} \equiv 2^0 2^{w \times 3} (\mathrm{mod} 19)$

5. Методом перебору знаходимо, що $w = 2$, тоді $x = 0 + 2 \times 3 = 6$.

Проаналізуємо четвертий крок ϱ-методу Поларда для дискретного логарифмування, розглянутого у пункті 1.2.5. Піднесення (1.4) до степеню λ дасть $a^d \equiv b^{\lambda n} (\mathrm{mod}\, p)$, де λn є формою dx. Якщо d буде дорівнювати 1,

то отримаємо, що λn є формою x. У такому випадку крок пошуку θ є надлишковим і можемо видати рішення задачі

$$x \equiv \log_a b \equiv \mu \times (v_i - v_{2i}) (\mathrm{mod}\, p-1),$$

перевіривши даний x на правильність.

2.2. Запропонована модифікація ϱ-методу Поларда для дискретного логарифмування

1. Розглянемо три числові послідовності $\{u_i\}$, $\{v_i\}$, $\{z_i\}$, $i = 1, 2, \ldots$, визначені наступним чином:

$$u_0 = v_0 = 0, \quad z_0 = 1;$$
$$\alpha = z_i (\mathrm{mod}\, 3);$$
$$u_{i+1} \equiv \begin{cases} u_i (\mathrm{mod}\, p-1), & \alpha = 0; \\ u_i + 1 (\mathrm{mod}\, p-1), & \alpha = 1; \\ 2u_i (\mathrm{mod}\, p-1), & \alpha = 2; \end{cases}$$
$$v_{i+1} \equiv \begin{cases} v_i + 1 (\mathrm{mod}\, p-1), & \alpha = 0; \\ v_i (\mathrm{mod}\, p-1), & \alpha = 1; \\ 2v_i (\mathrm{mod}\, p-1), & \alpha = 2; \end{cases}$$
$$z_{i+1} \equiv \begin{cases} z_i \times a (\mathrm{mod}\, p), & \alpha = 0; \\ z_i \times b (\mathrm{mod}\, p), & \alpha = 1; \\ z_i^{\,2} (\mathrm{mod}\, p), & \alpha = 2; \end{cases}$$

2. Розглядаємо набори $(z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$, $i = 1, 2, \ldots$, і шукаємо таке значення i, для якого $z_i = z_{2i}$, звідки

$$a^m \equiv b^n (\mathrm{mod}\, p),$$

де $m = u_i - u_{2i}$, $n = v_{2i} - v_i$.

3. Методом перебору для $x \in [1; p)$ знаходимо такий x, щоб

$$m \times x \equiv n (\mathrm{mod}\, p-1).$$

Розглянемо приклад вирішення порівняння

$$2^x \equiv 3 \ (\mathrm{mod}\, 23)$$

запропонованим модифікованим методом.

1. Розглянемо набори $(z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$, сформовані з трьох числових послідовностей $\{u_i\}$, $\{v_i\}$, $\{z_i\}$ (табл. 7).

Таблиця 7

Формування таблиці передобчислень для запропонованої модифікації ϱ-методу Поларда

i	u_i	v_i	z_i
0	0	0	1
1	1	0	3
2	1	1	6
3	1	2	12
4	1	3	1
5	2	3	3
6	2	4	6
7	2	5	12
8	2	6	1

2. $z_i = z_{2i} = 1$ досягається при $i = 4$. $m = u_4 - u_8 = -1 (\mathrm{mod}\, 22) = 21$,

$n = v_8 - v_4 = 3 (\mathrm{mod} 18) = 3$.

3. Методом перебору знаходимо, що $x = 19$:

$$21 \times 19 \equiv 399 \equiv 3 (\mathrm{mod}\, 22).$$

По-перше, модифікуємо перший крок ϱ-методу Поларда, використавши модулярну арифметику замість арифметики з плаваючою крапкою і тим самим позбуваємось необхідності у розрахунку граничних значень при розбитті на псевдовипадкові множини для знаходження проміжних допоміжних значень. У оригінальному методі граничні значення псевдовипадкових множин обчислюються шляхом ділення модуля p на три

частини, а у запропонованому – шляхом ділення поточного значення z за модулем 3, отримуючи три значення.

По-друге, модифікуємо останній крок методу. На цьому кроці у оригінальному методі перебираються можливі значення x з подальшим піднесенням до степеня, проте менше разів, ніж у методі перебору. Окрім цього у запропонованій модифікації маємо відсутність операції піднесення до степеня на останньому кроці.

2.3. Висновки

У першій частині поточного розділу було проаналізовано ϱ-метод Поларда для дискретного логарифмування, який є обраним для модифікації та наведено приклад, як можна вирішити задачу цим методом.

У другій частині поточного розділу наведено власну модифікацію проаналізованого вище методу. Проаналізовано суть та причину зміни ϱ-методу Поларда для дискретного логарифмування, а саме, по-перше, використання модулярної арифметики замість арифметики з плаваючою крапкою, оскільки позбуваємось необхідності у розрахунку граничних значень при розбитті на псевдовипадкові множини і, по-друге, позбуваємось необхідності у піднесенні до степеня на останньому кроці шляхом використання властивостей степенів.

3. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

3.1. Обґрунтування вибору мови програмування

Для розробки десктоп застосунку необхідно обрати мову програмування, бібліотеки для неї, а також IDE. Проведемо дослідження мов програмування, які використовуються для схожих цілей.

Нижче наведено табл. 2 рейтингів мов програмування. Серед мов програмування, опублікованих у рейтингах, у таблиці виключено ті мови, на яких розробка необхідного застосунку не може бути виконана – мови розмітки і мови для написання веб-застосунків.

Таблиця 2

Рейтинги мов програмування

Найвідоміші рейтинги мов програмування світу останніх років	Передумови створення рейтингу	Позиції у рейтингу мов програмування
Stack Overflow`s 2019 Developer Survey Results [5]	Створено на основі опитування 87 тис. респондентів – професіональних програмістів.	4. Python 5. Java 7. C# 10. C++
The PYPL PopularitY of Programming Language [6]	Створено на основі аналізу запитів у Google з пошуком керівництв на окремих мовах програмування.	1. Python 2. Java 4. C# 6. C/C++ 7. R 11. Matlab

Продовження табл. 2

The monthly TIOBE Programming Community Index for May 2020 [7]	Створено на основі результатів пошуку Google, Google Blogs, MSN, Yahoo!, Baidu, Wikipedia, що містять назву мови програмування. Рейтинг оновлюється раз на місяць.	1. C 2. Java 3. Python 4. C++ 5. C# 10. R 13. Matlab
The RedMonk Programming Language Rankings: January 2020 [8]	Створено на основі кореляції між використанням мови на GitHub та обговорень на Stack Overflow у вигляді графіку, тому нумерація неоднозначна.	2. Python 2. Java 5. C# 6. C++
IEEE Spectrum's 2019 ranking of top programming languages [9]	Створено на основі комбінації 11-ти метрик з 8-ми ресурсів – CareerBuilder, Google, GitHub, Hacker News, the IEEE, Reddit, Stack Overflow, і Twitter. Є поділ на 4 категорії, серед яких Enterprise (у тому числі десктоп-застосунки).	Серед Enterprise: 1. Python 2. Java 3. C 4. C++ 5. R 6. C# 7.Matlab

Бачимо, що мови програмування Python, Java, C#, C++, R, Matlab є найпопулярнішими. Дослідимо кожну з цих мов, аби обрати найкращу для написання десктоп застосунку, який би міг працювати з достатньо великими числами.

3.1.1. Python

Python – це інтерпретована високорівнева мова програмування загального призначення, яка широко відома мінімалістичним синтаксисом наряду з багатофункціональністю стандартної бібліотеки. Ця мова з’явилась порівняно нещодавно під впливом таких мов, як Lisp, Java, C++ і інших.

Python підтримує низку парадигм програмування, у тому числі ООП. Визначними рисами архітектури є динамічна типізація, автоматичне управління пам’яттю, підтримка багатопоточних обчислень і високорівневі структури даних.

Ця мова є найрозповсюдженішою у тому числі завдяки легкому синтаксису та великій кількості навчальних матеріалів і широко використовується і як інструмент для інтеграції та створення розширень, так і як основна мова. Частіше всього застосовується для створення веб-застосунків, для наукових обчислень замість мови Matlab, про яку йтиметься далі, для астрономічних розрахунків, в якості скриптової мови, графічного подання даних тощо. Однак Python має ряд недоліків:

- динамічна типізація, через яку ніколи не можна бути впевненим, на вхід функції йде число чи рядок, а також це збільшує використання пам’яті, тому що автоматично визначає тип змінної;
- низька швидкодія за рахунок інтерпретації коду замість попередньої компіляції;
- синтаксис і семантика мови часто можуть вводити в оману програмістів, що раніше мали справу з іншими мовами програмування – різна семантика присвоєння для вбудованих типів і об’єктів, нелогічна область видимості змінної у функції;
- непередбачувані помилки виконання, які часто знаходяться іншими мовами ще на етапі компіляції, викликають необхідність у дуже ретельному написанні коду і його тестуванні.

3.1.2. Java

Java – це об'єктно-орієнтована класозалежна мова загального призначення. Розробники мови притримувались гасла «пиши раз, запускай будь-де» (англ. Write once, run anywhere, WORA). Це значить, що скомпільований код Java може бути запущеним на будь-якій платформі, що підтримує Java без повторної компіляції. Синтаксис мови подібний

до C/C++, але має менше об’єктів низького рівня. Серед основних можливостей Java:

- автоматичне управління пам’яттю;
- набір можливостей для опрацювання виключень на помилок;
- багатий набір стандартних колекцій: список, стек, масив тощо;
- уніфікований доступ до баз даних;
- неявний спосіб перетворення стандартних типів;
- підтримка лямбд, загального типу даних та ін.

До недоліків мови можна віднести:

- авторське право, в результаті якого процес покращення мови є достатньо повільним в порівнянні з мовами відкритого доступу;
- підтримка загального типу даних (англ. generics) в Java підлягає серйозній критиці, адже реалізована за допомогою стирання типу, через що справжній тип параметра не є доступним під час виконання програми;
- відсутність реалізації беззнакових типів даних, що вдвічі зменшує граничні значення великих чисел, які часто використовуються у багатьох задачах числової обробки, у тому числі і криптографії, що робить Java програшною мовою у порівняння з іншими для таких цілей.

3.1.3. C#

C# – це об’єктно-орієнтована мова програмування, створена компанією Microsoft для розроблення застосунків для платформи Microsoft .NET Framework. По синтаксису мова найбільше схожа на C++ і Java, проте, створена пізніше, виключила неефективні та проблемні практики минулих мов.

C# має статичну типізацію з можливістю неявного оголошення і змінні загального (англ. generic) та пустого (англ. discard) типу, а також динамічне зв’язування. Це дозволяє як ширше використовувати методи без

необхідності дублювати код кілька разів задля передачі даних різного типу, так і не використовувати змінні там, де вони не потрібні. Мова також лишила підтримку поліморфізму, перевантаження методів, лямбда-вирази, що одночасно робить код ефективним та позбавляє потреби його дублювати. Успішно розроблені моделі делегатів і подій, які дозволяють не перевантажувати код безліччю зв'язків між класами. Також був доданий компонент LINQ (англ. Language Integrated Query – інтегровані в мову запити), який є не бібліотекою, а частиною мови і дозволяє виконувати запити схожі до SQL, щоб спрощувати обробку списків, баз, масивів тощо. Реалізація асинхронного та паралельного програмування також є дуже корисними можливостями мови.

Переваги мови C#:

- мова підтримує об'єктно-орієнтований підхід у тому числі у підході до реалізації абстрактних об'єктів з подальшою реалізацією взаємозв'язків між ними;
- синтаксичний цукор, представлений у мові, дозволяє позбуватись багатьох рядків коду;
- наявність великої кількості бібліотек, у тому числі для роботи з великими числами і розробкою десктоп-застосунків, дозволяє використовувати готові шаблони;
- велика кількість навчального матеріалу, а також зрозуміла документація;
- наявність строгої типізації, що дозволяє захиститись від небажаних викликів методів.

Серед недоліків мови C#:

- через велику кількість синтаксичного цукру у мові може втрачатись розуміння того, що робить код;
- код легко дизасемблювати, тому з великою долею ймовірності він може бути доступним «неправильним» людям;

- .NET використовує концепцію JIR-компіляції, тобто код буде скомпільованим тільки по мірі необхідності, що робить перший запуск доволі довгим.

3.1.4. C++

C++ – це об'єктно-орієнтована високорівнева мова загального призначення, розроблена як розширення мови програмування C. Ця мова значно розширилась і продовжує розвиватись. Наразі має об’єктно-орієнтовані, узагальнені і процедурні парадигми і низькорівневий доступ до пам’яті. Мова розроблялась як компільована мова і багато компаній надають компілятори C++, включаючи Oracle, Intel, Microsoft, IBM тощо, тому вона є доступною на багатьох платформах.

Спочатку мова призначалась для системного програмування і вбудованих систем, тому мала обмежені ресурси; хоча і знайшла своє використання у дуже широкому спектрі галузей, включаючи десктоп-застосунки, відеоігри, сервери тощо і славиться ефективністю та продуктивністю, проте має і ряд недоліків:

- ідеологія мови «програміст має контролювати все» часто призводить до значної кількості коду та складнощів у роботі з пам’яттю;
- відсутність параметричного поліморфізму, що призводить до оптимізації коду вручну замість використання перевантаження функцій, впливає на об’єм і складність коду;
- вбудовані засоби обходу обмежень є зайвими з міркувань безпеки;
- управління пам’яттю лежить на плечах програміста, що знижує ефективність роботи.

Наведені вище недоліки складають загальну складність і громіздкість мови.

3.1.5. Інші мови програмування

R – мова програмування для статистичної обробки даних і роботи з графікою. Ця мова широко застосовується для аналізу даних і статистики.

Зазвичай в R використовуються засоби командного рядка, хоч і є кілька GUI. Завдяки цій мові можна писати певні скрипти, за допомогою яких потім буде здійснюватись аналіз даних.

MATLAB – слаботипізована інтерпретована мова програмування високого рівня і включає інтегроване середовище розробки MatLab. Програми, створені за допомогою MatLab можуть бути двох типів – скрипти і функції, які не компілюються в машинний код і зберігаються у вигляді текстових документів.

Є і інші засоби для роботи з даними та математичними обчисленнями – ***Julia***, ***Fortran*** тощо. Вони також займають свою нішу серед мов програмування, орієнтованих на аналіз, проте як і усі у цьому підрозділі погано підходять для написання десктоп-застосунків.

3.1.6. Висновки

Внаслідок проведеного аналізу було встановлено, що мова програмування C# ефективно справиться з поставленими цілями, оскільки вона має ряд вбудованих бібліотек, що можуть працювати з різними математичними структурами, великими числами і є порівняно компактною для написання аналітичних програм.

3.2. Обґрунтування вибору бібліотек C#

Загалом мова програмування C# є достатньо самостійною для написання математичних функцій та алгоритмів. Завдяки їй можна розробити ефективні математичні моделі.

У даній дипломній роботі необхідно тестувати набори великих чисел. Для цього можна використати стандартний простір імен System.Numerics, у якому представлена структура даних BigInteger. Даний тип даних є незмінним, тобто при кожній зміні числа протягом виконання програми відбувається створення і повернення нового елементу. Через те, що BigInteger не має нижніх та верхніх меж, протягом виконання програми

числа можуть ставати неймовірно великими і цим сильно впливати на продуктивність застосунку.

Для написання графічного інтерфейсу існує кілька інструментів – UWP, WPF та WinForms. Проаналізуємо плюси і мінуси кожного, створених компанією Microsoft.

Universal Windows Platform (UWP) – платформа для шаблонування проектів, вперше введена в Windows 10. Ціль платформи – допомогти у розробці універсальних застосунків для Windows 10, Windows 10 Mobile, Xbox One і HoloLens. У цьому полягає основний недолік – вузьке коло застосування.

Windows Presentation Foundation (WPF) – це також графічна бібліотека з відкритими кодом, створена після проаналізованої нижче WinForms і під її впливом. Бібліотека дуже гнучка і у тому числі дозволяє краще розділяти дані та графічний інтерфейс.

Windows Forms (WinForms) – це графічна бібліотека з відкритим кодом, що забезпечує багатий набір інструментів для розробки клієнтських застосунків для персональних комп'ютерів та планшетів. Усі візуальні елементи наслідуються з головного класу Control, що несе за собою ефективний менеджмент наслідуваних елементів. Має велику кількість готових елементів управління. В порівнянні з WPF, про яку йтиметься далі, WinForms є більш старою і тому краще протестованою і перевіреною. Крім того, дизайнер Visual Studio більш пристосований до цієї бібліотеки, в порівнянні з тим, що для WPF більше необхідно робити програмісту, що могло бути зроблено програмним забезпеченням автоматично.

Враховуючи зручність і багатофункціональність бібліотеки WinForms разом зі зручною інтеграцією у найоптимальніший дизайнер Visual Studio, її було обрано для розроблення десктоп-застосунку даної дипломної роботи.

3.3. Висновки

У першій частині поточного розділу проаналізовано низку мов програмування (Python, Java, C#, C++ і інші), що зазвичай використовуються для програмування математичних моделей та методів. Крім того, зроблено порівняльну характеристику цих мов та внаслідок проведеного аналізу було обрано мову програмування C#, оскільки вона має найбільший набір бібліотек, що включають у себе необхідні для дослідження методи роботи з математичними структурами та великими числами.

У другій частині поточного розділу проаналізовано існуючі бібліотеки мови C# для розроблення графічного інтерфейсу, які теоретично могли би бути використані для виконання робіт. Проведено аналіз бібліотек для реалізації задачі дискретного логарифмування. В результаті аналізу обрано бібліотеку Windows Forms, яка є одним із найкращих апаратів для розроблення десктоп-застосунків від компанії Microsoft, тому що кожний компонент має багато властивостей і різних форм реагування на дії користувача.

Такий вибір мови програмування у поєднанні з бібліотекою для розроблення графічного інтерфейсу є найоптимальнішим для роботи над поставленою ціллю даної дипломної роботи.

4. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ МЕТОДІВ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

4.1. Розроблення десктоп-застосунку для аналізу методів дискретного логарифмування

Для дослідження методів дискретного логарифмування було створено десктоп-застосунок, який надає функціонал як калькулятора (рис. 4) для розв'язання задачі дискретного логарифмування, так і аналізатора (рис. 5) для дослідження часу роботи обраних на вкладці методів на різних за кількістю байт числах і з різною кількістю прогонів разів.

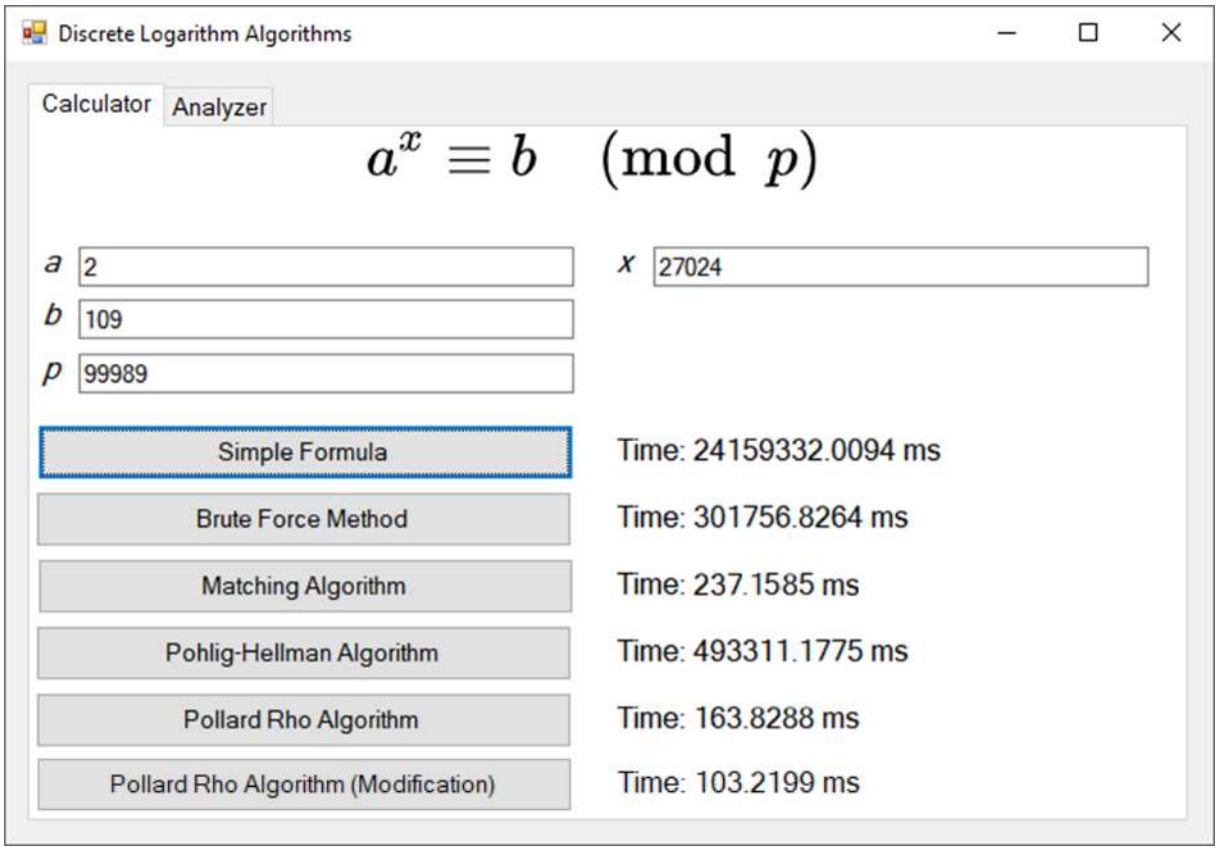


Рис. 4. Вигляд вкладки Калькулятор

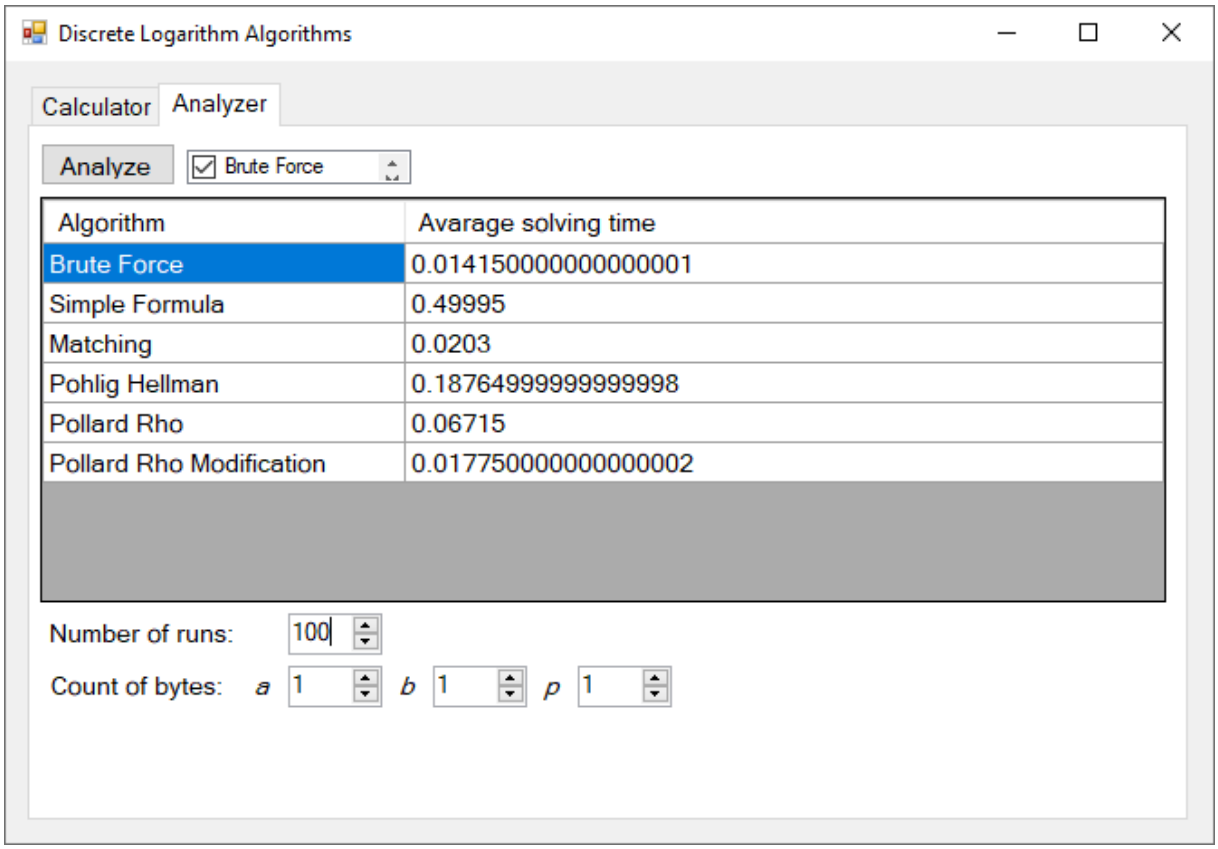


Рис. 5. Вигляд вкладки Аналізатор

Програмне забезпечення складається з frontend (клас MainForm і його часткові класи) і backend (класи Solver, Analyzer, BigMath) частин (рис. 6). Головна форма ділиться на два часткові класи – Калькулятор та Аналізатор головної форми, які взаємодіють з аналогічно названими класами для backend частини, розробленими шаблоном проектування Singleton.

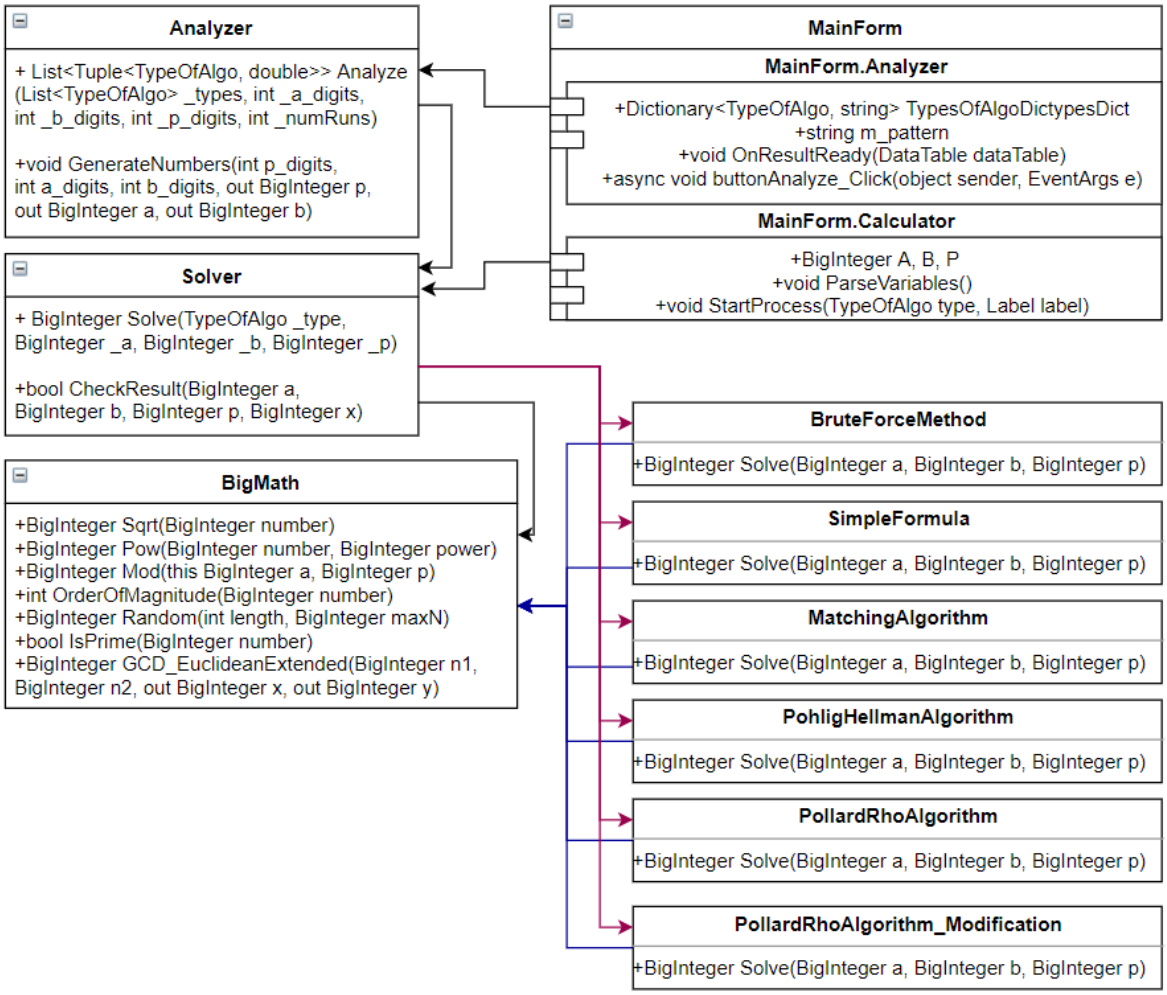


Рис. 6. Діаграма класів

Калькулятор (лістинг 1) має функції розв’язання задачі дискретного логарифмування шляхом виклику методу Solve статичного класу, обраного за допомогою переліку імен enum і перевірки правильності результату.

При аналізі часу роботи алгоритмів перший прогін необхідно не враховувати, тому що при першому виконанні методу виконується JIT-компіляція (англ. Just-in-time compilation, компіляція в цей же час) коду. Це означає, що байт-код буде скомпільований у машинний код тільки під час виконання програми, а отже займе більше часу, ніж без цього.

Лістинг 1. Методи класу Solver

```
private BigInteger PrivateSolve(TypeOfAlgo type, BigInteger a,
BigInteger b, BigInteger p)
{
    BigInteger result = -1

    switch (type)
    {
        case TypeOfAlgo.BruteForce:
            result = BruteForceMethod.Solve(a, b, p);
            break;
        case TypeOfAlgo.SimpleFormula:
            result = SimpleFormula.Solve(a, b, p);
            break;
        case TypeOfAlgo.Matching:
            result = MatchingAlgorithm.Solve(a, b, p);
            break;
        case TypeOfAlgo.PohligHellman:
            result = PohligHellmanAlgorithm.Solve(a, b, p);
            break;
        case TypeOfAlgo.PollardRho:
            result = PollardRhoAlgorithm.Solve(a, b, p);
            break;
        case TypeOfAlgo.PollardRhoModif:
            result = PollardRhoAlgorithm_Modification.Solve(a, b, p);
            break;

    return result;
}


public static bool CheckResult(BigInteger a, BigInteger b, BigInteger p,
BigInteger x)
{
    if (x < 0)
    {
        return false;
     }

    bool isSubstitutionCorrect = BigMath.Pow(a, x) % p == b;

    return isSubstitutionCorrect;
}
```

Аналізатор (лістинг 2) допомагає визначити середній час роботи необхідних методів, відкидаючи перший прогін, використовуючи калькулятор і генерацію псевдовипадкових чисел за допомогою криптографічного генератору випадкових чисел RNGCryptoServiceProvider з простору імен System.Security.Cryptography.

Лістинг 2. Методи класу Analyzer

```
private List<Tuple<TypeOfAlgo, double>> PrivateAnalyze(List<TypeOfAlgo>
types, int a_digits, int b_digits, int p_digits, int numRuns)
{
    List<Tuple<TypeOfAlgo, double>> result = new List<Tuple<TypeOfAlgo,
double>>()

    for (int iType = 0; iType < types.Count; iType++)
    {
        TypeOfAlgo type = types[iType];
        double allTime = 0;

        bool first = true; //bad result

        for (int iRun = 0; iRun < numRuns; iRun++)
        {
            GenerateNumbers(p_digits, a_digits, b_digits, out BigInteger p,
            out BigInteger a, out BigInteger b);

            var watch = System.Diagnostics.Stopwatch.StartNew();

            BigInteger solved = Solver.Solve(type, a, b, p);

            watch.Stop();

            if (Solver.CheckResult(a, b, p, solved))
            {
                allTime += first ? 0 : watch.Elapsed.TotalMilliseconds;
                first = false;
            }
            Else
            {
                iRun--;
            }
        }
        double averageTime = allTime / (numRuns - 1);
        result.Add(new Tuple<TypeOfAlgo, double>(type, averageTime));
    }

    return result;


private void GenerateNumbers(int p_digits, int a_digits, int b_digits, out
BigInteger p, out BigInteger a, out BigInteger b)
{
    do
    {
        p = BigMath.Random(p_digits);
    }
    while (!IsPPrime(p))

    do
```

```
    {
        a = BigMath.Random(a_digits);
    }

    while (!IsAGood(a, p))
    b = BigMath.Random(b_digits, p);
}
```

4.2. Програмна реалізація методів дискретного логарифмування та запропонованої модифікації ϱ-методу Поларда для дискретного логарифмування

Для програмної реалізації методів дискретного логарифмування був використаний підхід поєднання переліку типів алгоритмів enum з окремими класами, у яких імплементовано методи (лістинг 3).

Лістинг 3. Реалізація вибору методу вирішення

```
private BigInteger PrivateSolve(TypeOfAlgo type, BigInteger a, BigInteger
b, BigInteger p)

{
    BigInteger result = -1;


    switch (type)
    {
        case TypeOfAlgo.BruteForce:
            result = BruteForceMethod.Solve(a, b, p);
            break;
        case TypeOfAlgo.SimpleFormula:
            result = SimpleFormula.Solve(a, b, p);
            break;
        case TypeOfAlgo.Matching:
            result = MatchingAlgorithm.Solve(a, b, p);
            break;
        case TypeOfAlgo.PohligHellman:
            result = PohligHellmanAlgorithm.Solve(a, b, p);
            break;
        case TypeOfAlgo.PollardRho:
            result = PollardRhoAlgorithm.Solve(a, b, p);
            break;
```

```
        case TypeOfAlgo.PollardRhoModif:
            result = PollardRhoAlgorithm_Modification.Solve(a, b, p);
            break;
    return result;
}
```

У лістингу 4 наведено програмну реалізацію класичного ϱ-методу Поларда.

Лістинг 4. Клас реалізації ϱ-методу Поларда

```
class PollardRhoAlgorithm
{
    public static BigInteger Solve(BigInteger _a, BigInteger _b,
BigInteger p)
    {
        BigInteger r = _a, q = _b;
        BigInteger x = 1, a = 0, b = 0;
        BigInteger X = x, A = a, B = b;

        for (int iterator = 1; iterator < p; iterator++)
        {
            RefreshValues(ref x, ref a, ref b, r, q, p);
            RefreshValues(ref X, ref A, ref B, r, q, p); //2i - 2
            RefreshValues(ref X, ref A, ref B, r, q, p);
            if (x == X)
            {
                BigInteger m = (a - A).Mod(p - 1),
                    n = (B - b).Mod(p - 1);

                if (m == 0)
                {
                    return -1;
                }

                BigInteger gcd = BigMath.GCD_EuclideanExtended(m, p - 1,
out BigInteger mu, out BigInteger pu);
```

```
                BigInteger temp =  (mu * n).Mod(p - 1);


                for (BigInteger w = 0; w <= gcd; w++)
                {
                    BigInteger result = ((temp + w * (p - 1)) / gcd)
.Mod(p - 1);
                    if (BigMath.Pow(r, result) % p == q)
                    {
                        return result;
                    }
                    else if (w % 2 == 0 &&
BigMath.Pow(-r, result).Mod(p == q)
                    {
                        return result;
                    }
                }
                return -1;
            }
        }
        return -1;
    }



    private static void RefreshValues(ref BigInteger x, ref BigInteger a,
ref BigInteger b,
        BigInteger r, BigInteger q, BigInteger p)
    {
        if ((0 <= x) && (x <= p / 3))
        {
            x = q * x;
            a++;
        }
        else if ((p / 3 < x) && (x <= 2 * p / 3))
        {
            x = x * x;
            a = 2 * a;
            b = 2 * b;
```

```
        }
        else if ((2 * p / 3 < x) && (x <= p))
        {
            x = r * x;
            b++;
        }

        x = x % (p);
        a = a % (p - 1);
        b = b % (p - 1);
    }
}
```

У лістингу 5 наведено реалізацію запропонованої модифікації ϱ-методу Поларда.

Лістинг 5. Клас реалізації запропонованої модифікації ϱ-методу Поларда

```
class PollardRhoAlgorithm_Modification
{
    public static BigInteger Solve(BigInteger _a, BigInteger _b,
BigInteger p)
    {
        BigInteger r = _a, q = _b;
        BigInteger x = 1, a = 0, b = 0;
        BigInteger X = x, A = a, B = b

        for (int iterator = 1; iterator < p; iterator++)
        {
            RefreshValues(ref x, ref a, ref b, r, q, p);
            RefreshValues(ref X, ref A, ref B, r, q, p); //2i - 2
            RefreshValues(ref X, ref A, ref B, r, q, p);

            if (x == X)
            {
                BigInteger m = (a - A).Mod(p - 1),
```

```
                n = (B - b).Mod(p - 1);

            if (m == 0)
            {
                return -1;
            }

            for (BigInteger i = 1; i < p; i++)
            {
                BigInteger temp = m * i % (p - 1);
                if (temp == n)
                {
                    return i;
                }
            return -1;
        }
    return -1;
}
```

```
private static void RefreshValues(ref BigInteger x, ref BigInteger a, ref
BigInteger b, BigInteger r, BigInteger q, BigInteger p)
{
    int i = (int)x % 3;
    switch (i)
    {
        case 0:
            x = r * x;
            b++;
            break;
        case 1:
            x = q * x;
            a++;
            break;
        case 2:
            x = x * x;
            a = 2 * a;
            b = 2 * b;
```

```
                break;
            default:
                break;

        x = x % (p);
        a = a % (p - 1);
        b = b % (p - 1);
    }
}
```

4.3. Порівняльний аналіз швидкодії запропонованого методу дискретного логарифмування з існуючими

Для порівняльного аналізу швидкодії методів дискретного логарифмування було використано розроблене програмне забезпечення. Під час аналізу проводились заміри часу виконання методів для різної кількості прогонів і на різних за кількістю байт числах.

На рис. 4 наведений приклад замірів часу роботи алгоритмів за достатньо великим модулем p. Практичною формулою рішення досягається за майже 7 год, алгоритмом Поліга-Гелмана – за 8.5 хв, методом перебору – за 5 хвилин, алгоритмом узгодження – за 0.2 сек, ϱ-методом Поларда – за 0.16 сек і запропонованою модифікацією ϱ-метода Поларда – за 0.1 сек.

При аналізі часу роботи методів дискретного логарифмування було встановлено, що метод перебору є найповільнішим (рис. 7). Кожній точці на графіку відповідає 10 прогонів відповідного алгоритму. Через те, що на графіку важко порівняти інші методи, метод перебору було вилучено зі статистики.

Другим по величині часу роботи виявився метод розв’язання за допомогою формули, виведеної практичним шляхом (рис. 8). Через аналогічну ситуацію з неможливістю візуально порівняти методи, вирішення за формулою також було вилучено зі статистики.

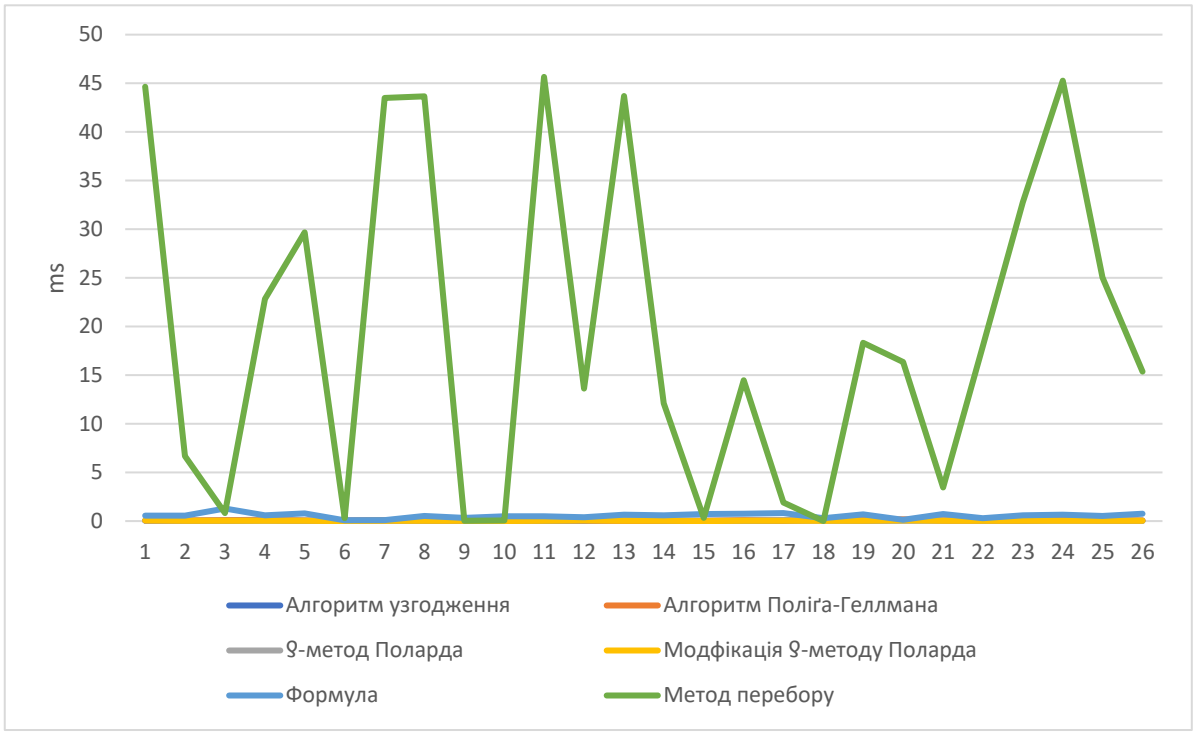


Рис. 7. Час роботи всіх описаних у роботі методів дискретного логарифмування

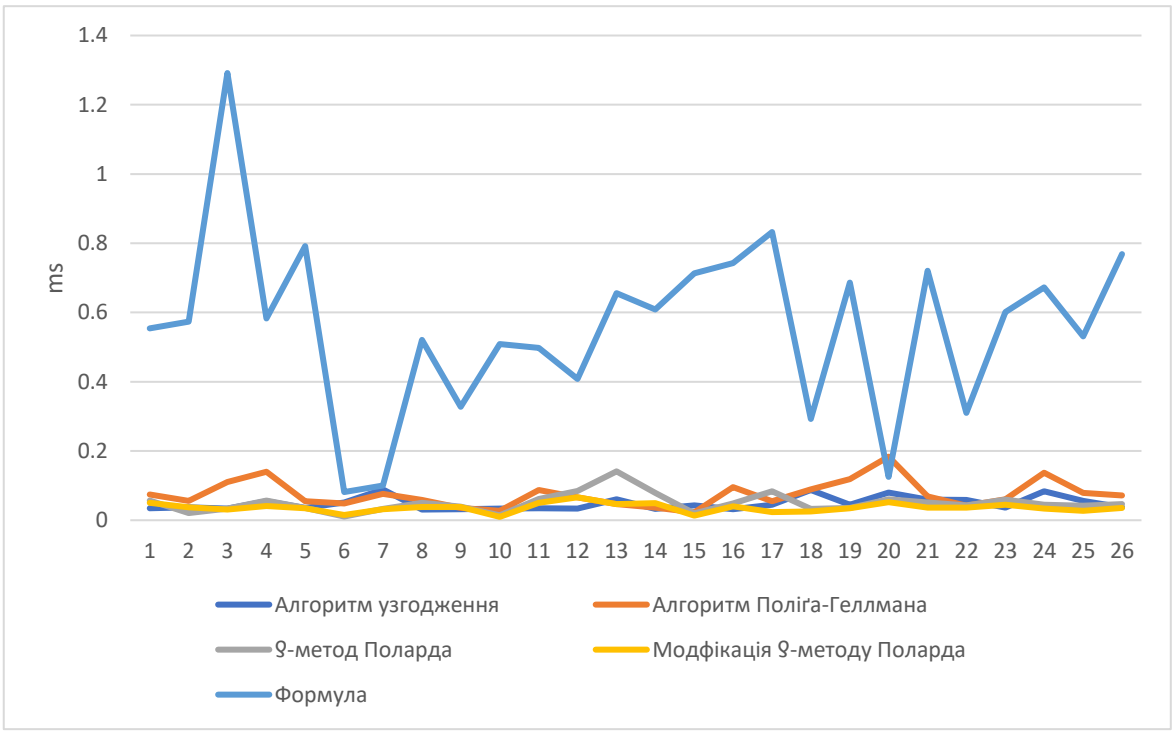


Рис. 8. Час роботи всіх (крім методу перебору) описаних у роботі методів дискретного логарифмування

Як і очікувалось в теорії, на практиці формула і метод перебору показують найгірші результати, а модифікація ϱ-методу Поларда – найкращі (рис. 9). Модифікований метод значно пришвидшує пошук роз'вязку дискретного логарифму, що визначено для простого модуля p довжиною 2 байти.

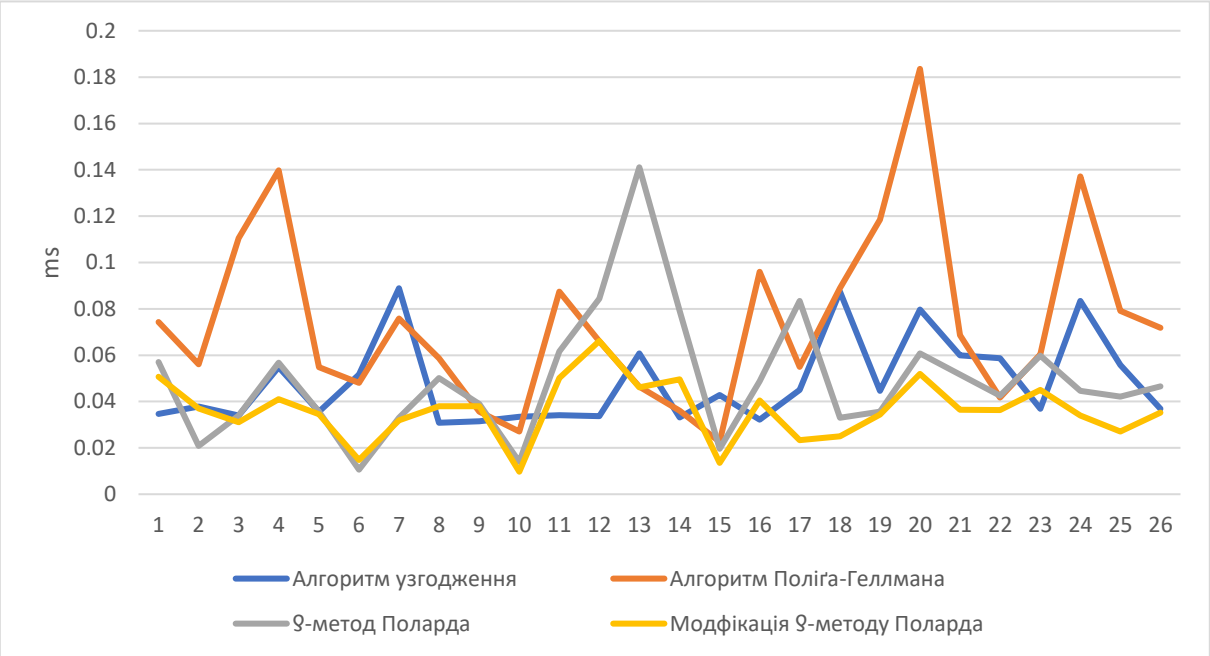


Рис. 9. Час роботи всіх (крім методу перебору і формули) описаних у роботі методів дискретного логарифмування

На рис. 10 наведено порівняльний графік часу роботи ϱ-методу Поларда та запропонованої модифікації методу для $\boldsymbol{p}$ довжиною 2 байти. Проаналізувавши графік, час роботи запропонованої модифікації покращує класичний ϱ-метод Поларда.

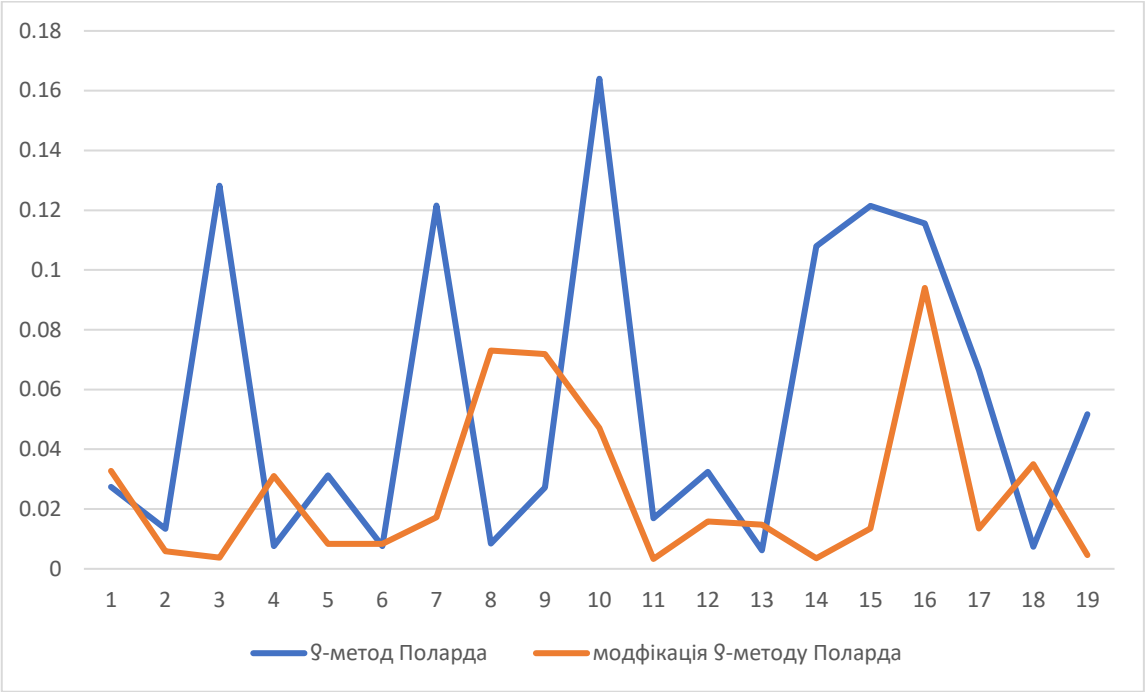


Рис. 10. Час роботи ϱ-методу Поларда та запропонованої модифікації методу дискретного логарифмування для ***p*** довжиною 2 байти

Експериментальні дослідження проводились на комп'ютері MSI GL-65 95C на операційній системі Windows 10 з наступними характеристиками: CPU 2.6 ГГц, Intel Core i7, оперативна пам'ять 16 Гб.

4.4. Висновки

У першій частині поточного розділу продемонстровано розроблене програмне забезпечення, лістинги частини функцій, а також аналіз діаграми класів застосунку.

У другій частині поточного розділу проаналізовано програмну реалізацію методів дискретного логарифмування, наведених у даній дипломній роботі і наведено лістинги реалізованих методів.

У третій частині поточного розділу виконано дослідження швидкодії методів дискретного логарифмування і показано, що запропонована модифікація ϱ-методу Поларда має у 1.5 рази менший час роботи ніж у класичного методу.

ВИСНОВКИ

Підсумовуючи результати виконання даної дипломної роботи, можна зробити наступні висновки:

1. Проведено аналіз таких методів дискретного логарифмування, як метод перебору, формула, виведена практичним шляхом, алгоритм узгодження, алгоритм Поліга-Гелмана, ϱ-метод Поларда.
2. Запропоновано модифікацію ϱ-метода Поларда, яка відрізняється від існуючого тим, що при розбитті на псевдовипадкові множини використовується модулярна арифметика замість арифметики з плаваючою крапкою, а також відсутністю операції піднесення до степені на останньому кроці, оскільки це забезпечує полегшення обрахувань.
3. Проведено аналіз низки мов для програмування математичних моделей та методів. Внаслідок проведеного аналізу обрано мову програмування C#, оскільки вона має найбільший набір бібліотек, що включають у себе необхідні для дослідження методи роботи з математичними структурами та великими числами, а також графічну бібліотеку Windows Forms для розроблення десктоп-застосунків. Крім того, обрано дизайнер Visual Studio, тому що він найбільше пристосований до роботи з бібліотекою WinForms.
4. Розроблено програмне забезпечення для дослідження методів дискретного логарифмування, а також протестовано розроблене програмне забезпечення.
5. Проведено експериментальні дослідження часу роботи алгоритмів, які показали зменшення часу роботи для модифікації ϱ-методу Поларда в 1.5 рази порівняно з класичним методом для простого модуля довжиною 2 байти.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Василенко, О. Н. Теоретико-числовые алгоритмы в криптографии [Текст] / О. Н. Василенко. — Москва: МЦНМО, 2003. — 328 с.
2. Odlyzko, A. M. Discrete logarithms in finite fields and their cryptographic significance [Text] / A. M. Odlyzko. — Murray Hill, New Jersey, 1984. — 6 p.
3. Pohlig, S. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance [Text] / S. Pohlig, M. Hellman // IEEE Trans. Inform. Theory. — 1978. — Vol. 24. — 106-110 p. — ISSN: 0018-9448
4. Pollard, J.M. A Monte Carlo methods for index computation (mod p) [Text] / J.M. Pollard // Math. Comp. — 1978. — Vol. 32. — 918-924 p.
5. StackOverflow`s Developer Survey Results 2019 [Електронний ресурс] — Режим доступу: https://insights.stackoverflow.com/survey/2019#technology-_-programming-scripting-and-markup-languages
6. PYPL PopularitY of Programming Language [Електронний ресурс] — Режим доступу: http://pypl.github.io/PYPL.html
7. TIOBE Index for May 2020 [Електронний ресурс] — Режим доступу: https://www.tiobe.com/tiobe-index/
8. The RedMonk Programming Language Rankings: January 2020 [Електронний ресурс] — Режим доступу: https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/
9. The Top Programming Languages 2019 [Електронний ресурс] — Режим доступу: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019

ДОДАТКИ

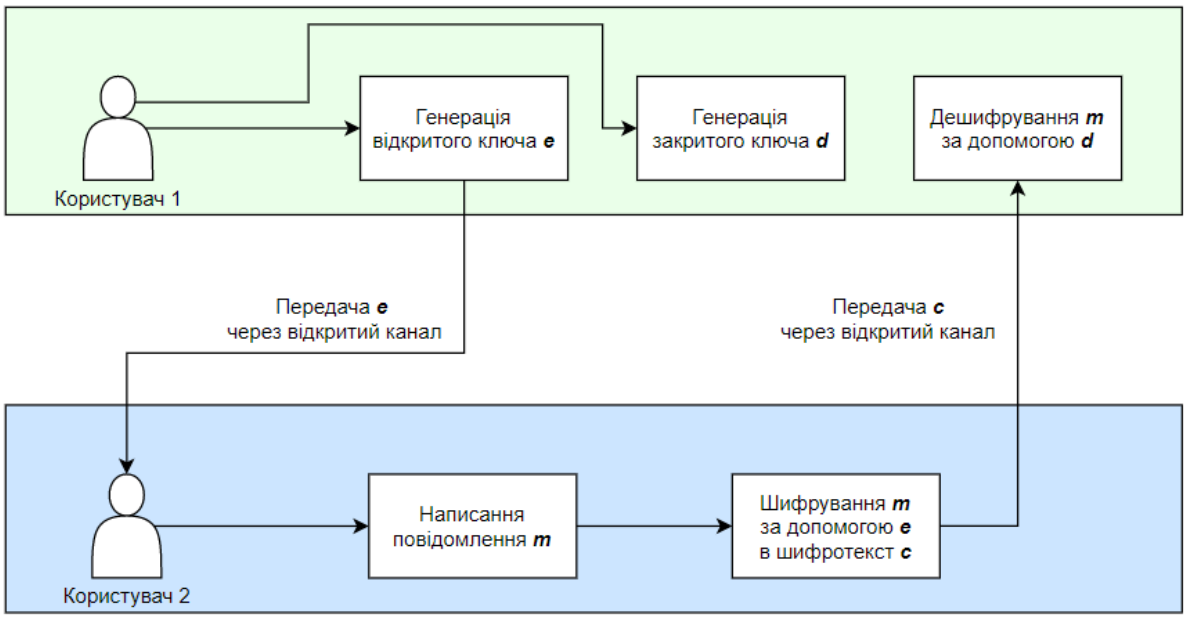
Додаток 1

Копії графічних матеріалів

ПРИНЦИП РОБОТИ ВІДКРИТОГО ТА ЗАКРИТОГО КЛЮЧІВ

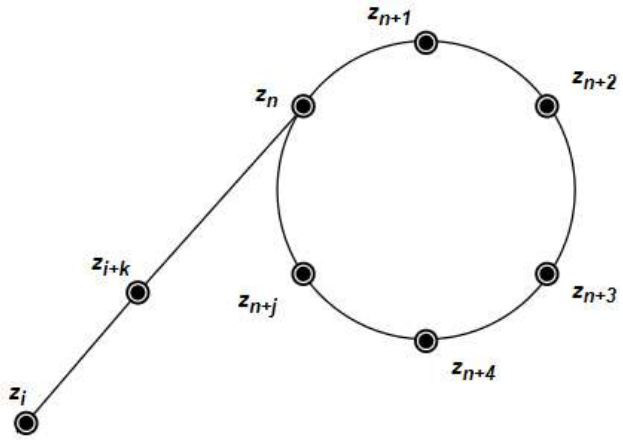
Мірошник Віталіна Ігорівна, гр. КП-62

ПРИНЦИП РОБОТИ АСИМЕТРИЧНОЇ КРИПТОСИСТЕМИ



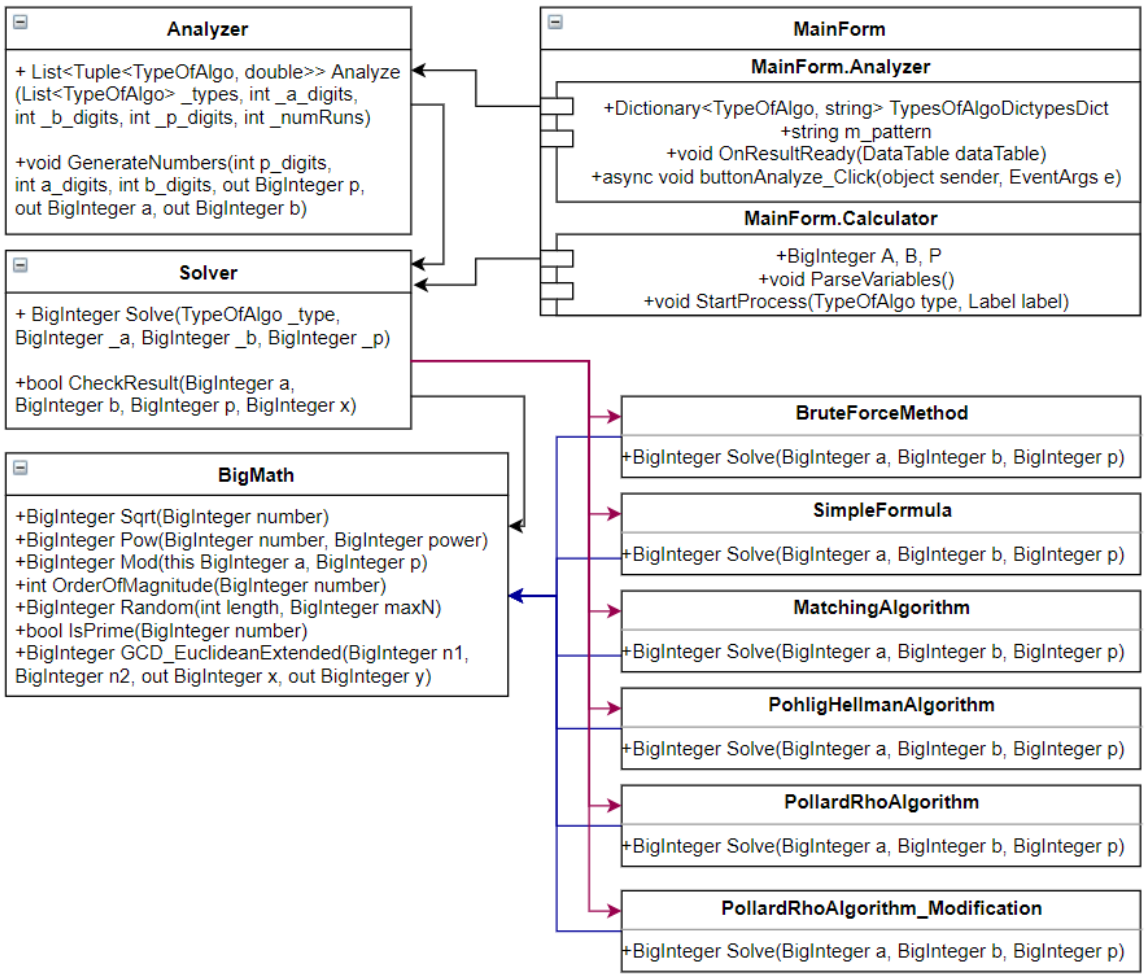
Мірошник Віталіна Ігорівна, гр. КП-62

Схематичне зображення точок послідовності ϱ-методу Поларда



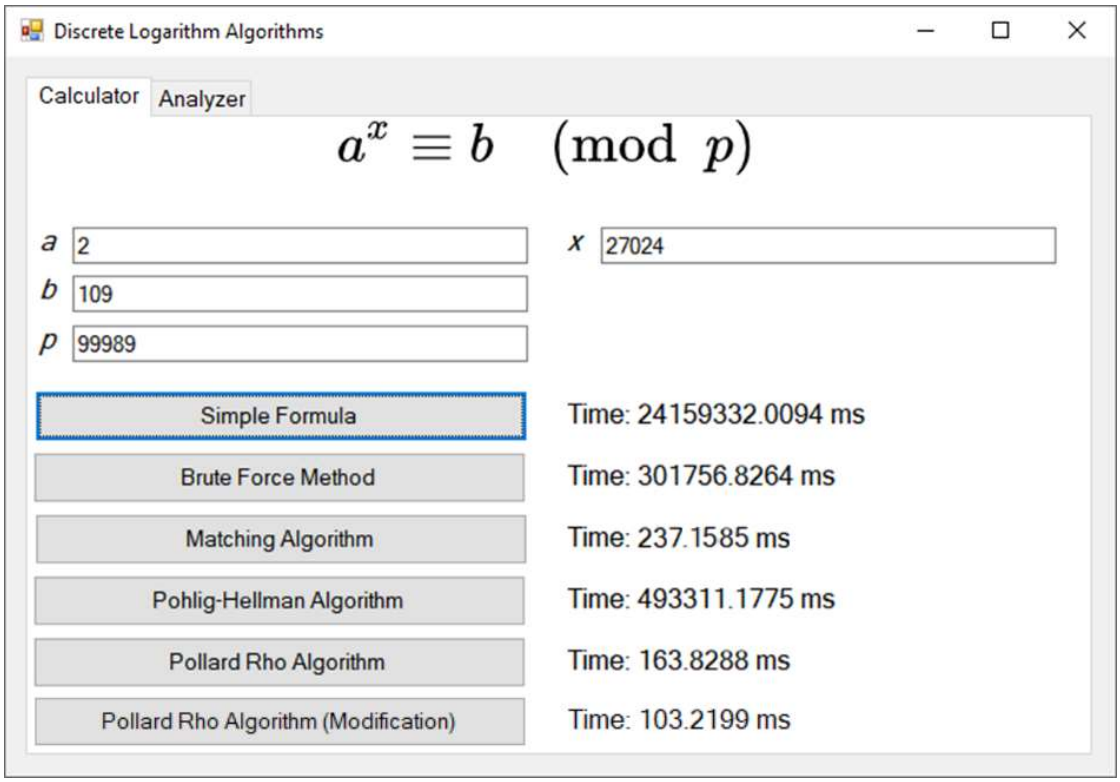
Мірошник Віталіна Ігорівна, гр. КП-62

Діаграма класів



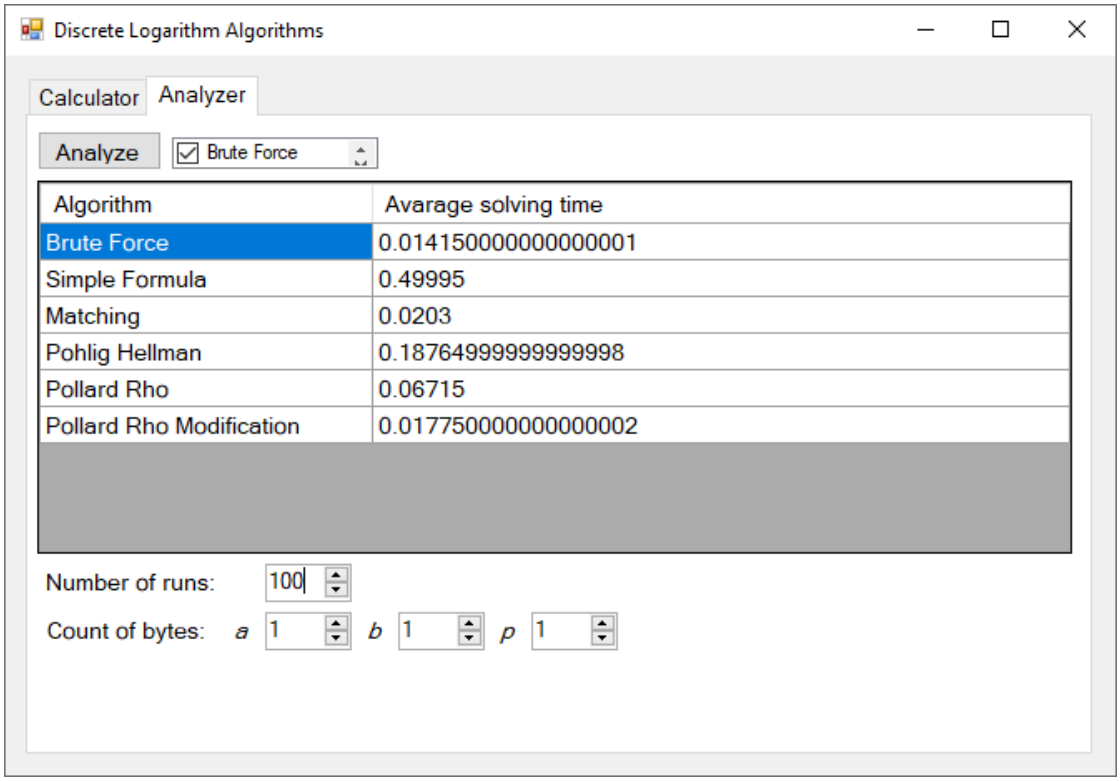
Мірошник Віталіна Ігорівна, гр. КП-62

Вигляд вкладки Калькулятор



Мірошник Віталіна Ігорівна, гр. КП-62

Вигляд вкладки Аналізатор

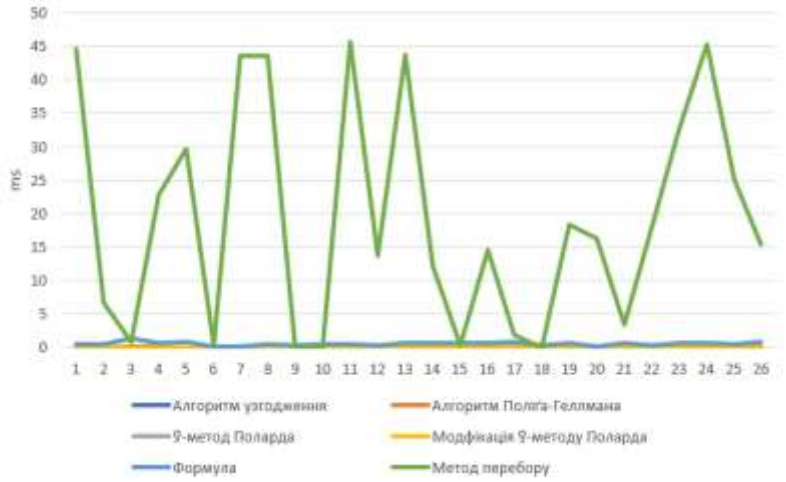


Мірошник Віталіна Ігорівна, гр. КП-62

Час роботи всіх (крім методу перебору і формули) описаних у роботі методів дискретного логарифмування

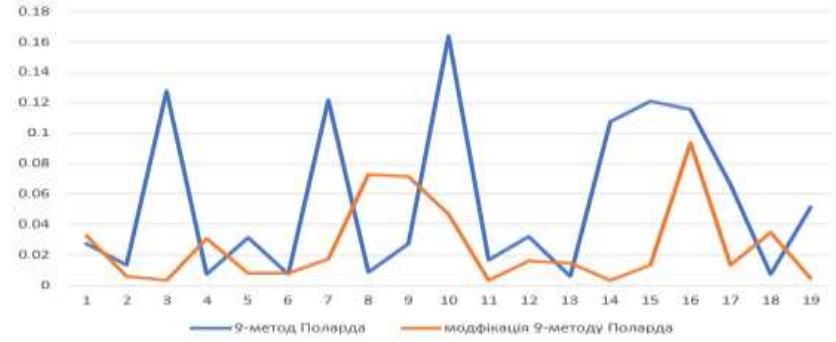
Мірошник Віталіна Ігорівна, гр. КП-62

Час роботи всіх описаних у роботі методів дискретного логарифмування



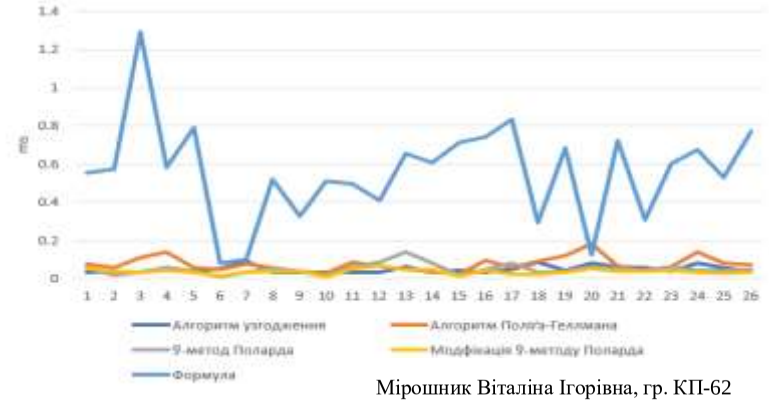
Мірошник Віталіна Ігорівна, гр. КП-62

Час роботи ϱ-методу Поларда та запропонованої модифікації методу дискретного логарифмування для p довжиною 2 байти



Мірошник Віталіна Ігорівна, гр. КП-62

Час роботи всіх (крім методу перебору) описаних у роботі методів дискретного логарифмування



Мірошник Віталіна Ігорівна, гр. КП-62

Додаток 2

Текст програми

```
namespace discrete_logarithm_algorithms
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }

    }
}


public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
            InitializeCalculator();
            InitializeAnalyzer();
        }


        private void textBox_TextChanged(object sender, EventArgs e)
        {
            if (sender is TextBox textBox)
            {
                bool isEverythingGood = BigInteger.TryParse(textBox.Text,
out BigInteger number);

                textBox.BackColor = isEverythingGood ?
                    SystemColors.Window : Color.FromArgb(255, 114, 111);
            }
        }
```

```
    }
partial class MainForm //Calculator Tab
    {
        BigInteger A, B, P;

        private void InitializeCalculator()
        {
            A = 0;
            B = 0;
            P = 0;

            textBoxA.Text = "3";
            textBoxB.Text = "13";
            textBoxP.Text = "17";
            ParseVariables();
        }

        private void ParseVariables()
        {
            A = BigInteger.Parse(textBoxA.Text);
            B = BigInteger.Parse(textBoxB.Text);
            P = BigInteger.Parse(textBoxP.Text);
        }

        private void buttonSimpleFormula_Click(object sender, EventArgs e)
        {
            StartProcess(TypeOfAlgo.SimpleFormula, labelSF);
        }

        private void buttonBruteForce_Click(object sender, EventArgs e)
        {
            StartProcess(TypeOfAlgo.BruteForce, labelBF);
        }

        private void buttonMatching_Click(object sender, EventArgs e)
        {
```

```
            StartProcess(TypeOfAlgo.Matching, labelMA);
        }

        private void buttonPohligHellman_Click(object sender, EventArgs e)
        {
            StartProcess(TypeOfAlgo.PohligHellman, labelPHA);
        }

        private void buttonRhoPollard_Click(object sender, EventArgs e)
        {
            StartProcess(TypeOfAlgo.PollardRho, labelRho);
        }

        private void buttonModif_Click(object sender, EventArgs e)
        {
            StartProcess(TypeOfAlgo.PollardRhoModif, labelModif);
        }

        private void StartProcess(TypeOfAlgo type, Label label)
        {
            ParseVariables();
            var watch = System.Diagnostics.Stopwatch.StartNew();
            BigInteger result = Solver.Solve(type, A, B, P);
            watch.Stop();
            double elapsedMs = watch.Elapsed.TotalMilliseconds;
            textBoxX.Text = result.ToString();
            label.Text = "Time: " + elapsedMs.ToString() + " ms";
        }
    }
partial class MainForm //Analyzer Tab
    {
        delegate void resultHandler(DataTable dt);
        private event resultHandler ResultReady;

        private const string AlgorithmCol = "Algorithm";
        private const string AvarageCol = "Avarage solving time";
        readonly int PATTERN_LENGTH = TypesOfAlgoDictypesDict.Count;
```

```
        string m_pattern = "111111";
        static readonly Dictionary<TypeOfAlgo, string>
TypesOfAlgoDictypesDict = new Dictionary<TypeOfAlgo, string>()
        {
            { TypeOfAlgo.BruteForce,
                "Brute Force" },
            { TypeOfAlgo.SimpleFormula,
                "Simple Formula" },
            { TypeOfAlgo.Matching,
                "Matching" },
            { TypeOfAlgo.PohligHellman,
                "Pohlig Hellman" },
            { TypeOfAlgo.PollardRho,
                "Pollard Rho" },
            { TypeOfAlgo.PollardRhoModif,
                "Pollard Rho Modification" }
        };

        private void InitializeAnalyzer()
        {
            for (int i = 0; i < PATTERN_LENGTH; i++)
            {
                checkedListBox.SetItemChecked(i, m_pattern[i] == '1');
            }

            ResultReady += OnResultReady;
        }

        internal void OnResultReady(DataTable dataTable)
        {
            if (InvokeRequired)
            {
                Invoke(new resultHandler(OnResultReady), new object[] {
dataTable });
            }
            else
            {
                ChangeDataSourceOfGrid(dataTable);
```

```
            }
        }

        public void ChangeDataSourceOfGrid(DataTable dataTable)
        {
            gridMain.DataSource = null;
            gridMain.DataSource = dataTable;
            SetColumnWidth();
        }

        private void SetColumnWidth()
        {
            if (gridMain.Columns.Contains(AlgorithmCol))
            {
                gridMain.Columns[AlgorithmCol].Width = 200;
            }

            if (gridMain.Columns.Contains(AvarageCol))
            {
                gridMain.Columns[AvarageCol].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
            }
        }

        private void checkedListBox_MouseEnter(object sender, EventArgs e)
        {
            if (sender is CheckedListBox listBox)
            {
                int itemHeight = 16;
                listBox.Height = itemHeight * PATTERN_LENGTH;
            }
        }

        private void checkedListBox_MouseLeave(object sender, EventArgs e)
        {
            if (sender is CheckedListBox listBox)
            {
                listBox.Height = 19;
```

```
            }
        }

        private async void buttonAnalyze_Click(object sender, EventArgs e)
        {
            SetAlgoPatternFromUI();
            if (!m_pattern.Contains("1"))
            {
                MessageBox.Show("Please select at least one column to
display!", "Error");
                return;
            }

            List<TypeOfAlgo> neededTypes = new List<TypeOfAlgo>();
            for (int i = 0; i < m_pattern.Length; i++)
            {
                if (m_pattern[i] == '1')
                {
                    neededTypes.Add((TypeOfAlgo)i);
                }
            }

            progressBar.Visible = true;
            buttonAnalyze.Enabled = false;

            await Task.Factory.StartNew(() =>
            {
                List<Tuple<TypeOfAlgo, double>> resultList =
Analyzer.Analyze(neededTypes,
                    (int)numericUpDownA.Value, (int)numericUpDownB.Value,
(int)numericUpDownP.Value, //a, b, p bytes
                    (int)numericNumRuns.Value); //number of runs

                DataTable dt = ToDataTable(resultList);

                ResultReady?.Invoke(dt);
            });

            progressBar.Visible = false;
```

```
            buttonAnalyze.Enabled = true;
        }

        private DataTable ToDataTable(List<Tuple<TypeOfAlgo, double>>
items)
        {
            DataTable dataTable = new DataTable("Results");

            dataTable.Columns.Add(AlgorithmCol, typeof(string));
            dataTable.Columns.Add(AvarageCol, typeof(double));

            foreach (var item in items)
            {
                var values = new object[dataTable.Columns.Count];
                values[0] = TypesOfAlgoDictypesDict[item.Item1];
                values[1] = item.Item2;

                dataTable.Rows.Add(values);
            }

            return dataTable;
        }

        void SetAlgoPatternFromUI()
        {
            m_pattern = "";
            for (int i = 0; i < checkedListBox.Items.Count; i++)
            {
                if (checkedListBox.GetItemChecked(i))
                    m_pattern += "1";
                else
                    m_pattern += "0";
            }
        }
    }
class Analyzer
    {
```

```
        public static List<Tuple<TypeOfAlgo, double>>
Analyze(List<TypeOfAlgo> _types, int _a_digits, int _b_digits, int
_p_digits, int _numRuns)
        {
            return Instance.PrivateAnalyze(_types, _a_digits, _b_digits,
_p_digits, _numRuns);
        }


        private List<Tuple<TypeOfAlgo, double>>
PrivateAnalyze(List<TypeOfAlgo> types, int a_digits, int b_digits, int
p_digits, int numRuns)
        {
            List<Tuple<TypeOfAlgo, double>> result = new
List<Tuple<TypeOfAlgo, double>>();


            for (int iType = 0; iType < types.Count; iType++)
            {
                TypeOfAlgo type = types[iType];
                double[] allTimeForRuns = new double[types.Count];
                bool first = true; //bad result
                for (int iRun = 0; iRun < numRuns; iRun++)
                {
                    GenerateNumbers(p_digits, a_digits, b_digits,
                        out BigInteger p, out BigInteger a, out BigInteger
b);


                    var watch = System.Diagnostics.Stopwatch.StartNew();
                    BigInteger solved = Solver.Solve(type, a, b, p);
                    watch.Stop();
                    if (Solver.CheckResult(a, b, p, solved))
                    {
                        allTimeForRuns[iType] += first ? 0 :
watch.Elapsed.TotalMilliseconds;
                        first = false;
                    }
                    else
                        iRun--;
                }
                double averageTime = allTimeForRuns[iType] / (numRuns - 1);
                result.Add(new Tuple<TypeOfAlgo, double>(type,
averageTime));             }
```

```
            return result;
        }

        private void GenerateNumbers(int p_digits, int a_digits, int
b_digits, out BigInteger p, out BigInteger a, out BigInteger b)
        {
            do
            {
                p = BigMath.Random(p_digits);
            }
            while (!IsPGood(p));

            do
            {
                a = BigMath.Random(a_digits);
            }
            while (!IsAGood(a, p));

            b = BigMath.Random(b_digits, p);
        }

        private bool IsAGood(BigInteger a, BigInteger p)
        {
            return a > 1 && BigMath.OrderOfMagnitude(a) <=
BigMath.OrderOfMagnitude(p - 1);
        }

        private bool IsPGood(BigInteger p) //needs to be prime
        {
            return p > 2 && BigMath.IsPrime(p);
        }

        private Analyzer() //needs to be private to construct only in
Instance
        {
        }

        private static Analyzer Instance
        {
```

```
            get
            {
                return Nested.instance;
            }
        }

        private class Nested //instantiated lazily, because not marked with
flag "BeforeFieldInit"
        {
            static Nested() //static ctor -> not marked with flag
"BeforeFieldInit"
            {
            }

            internal static readonly Analyzer instance = new Analyzer();
//instantiated once when it's first referenced
        }
    }
class Solver
    {
        public static BigInteger Solve(TypeOfAlgo _type, BigInteger _a,
BigInteger _b, BigInteger _p)
        {
            return Instance.PrivateSolve(_type, _a, _b, _p);
        }

        private BigInteger PrivateSolve(TypeOfAlgo type, BigInteger a,
BigInteger b, BigInteger p)
        {
            BigInteger result = -1;

            switch (type)
            {
                case TypeOfAlgo.BruteForce:
                    result = BruteForceMethod.Solve(a, b, p);
                    break;
                case TypeOfAlgo.SimpleFormula:
                    result = SimpleFormula.Solve(a, b, p);
                    break;
                case TypeOfAlgo.Matching:
```

```
                    result = MatchingAlgorithm.Solve(a, b, p);
                    break;
                case TypeOfAlgo.PohligHellman:
                    result = PohligHellmanAlgorithm.Solve(a, b, p);
                    break;
                case TypeOfAlgo.PollardRho:
                    result = PollardRhoAlgorithm.Solve(a, b, p);
                    break;
                case TypeOfAlgo.PollardRhoModif:
                    result = PollardRhoAlgorithm_Modification.Solve(a, b,
p);
                    break;
            }

            Console.WriteLine(CheckResult(a, b, p, result) ?
"_____________________correct" : "wrong");

            return result;
        }
        public static bool CheckResult(BigInteger a, BigInteger b,
BigInteger p, BigInteger x)
        {
            if (x < 0)
            {
                return false;
            }

            bool isSubstitutionCorrect = BigMath.Pow(a, x) % p == b;
            return isSubstitutionCorrect;
        }

        private Solver() //needs to be private to construct only in
Instance
        {
        }

        private static Solver Instance
        {
            get
```

```
            {
                return Nested.instance;
            }
        }

        private class Nested //instantiated lazily, because not marked with
flag "BeforeFieldInit"
        {
            static Nested() //static ctor -> not marked with flag
"BeforeFieldInit"
            {
            }

            internal static readonly Solver instance = new Solver();
//instantiated once when it's first referenced
        }
    }
class BruteForceMethod
    {
        public static BigInteger Solve(BigInteger a, BigInteger b,
BigInteger p)
        {
            BigInteger lastIteration = p*p;

            BigInteger result = a;
            for (BigInteger i = 1; i <= lastIteration; i++)
            {
                if (result % p == b)
                {
                    return i;
                }

                result *= a;
            }

            return -1;
        }
    }
class MatchingAlgorithm
```

```
    {
        public static BigInteger Solve(BigInteger a, BigInteger b,
BigInteger p)
        {
            BigInteger H = BigMath.Sqrt(p) + 1;


            //c = a^H mod p
            BigInteger c = BigMath.Pow(a, H);
            c = c % p;
            Dictionary<BigInteger, BigInteger> table1 = new
Dictionary<BigInteger, BigInteger>();
            for (BigInteger u = 1; u <= H; u++)
            {
                table1.Add(u, BigMath.Pow(c, u) % p);
            }


            Dictionary<BigInteger, BigInteger> table2 = new
Dictionary<BigInteger, BigInteger>();
            for (BigInteger v = 0; v <= H; v++)
            {
                table2.Add(v, b * BigMath.Pow(a, v) % p);
                foreach (KeyValuePair<BigInteger, BigInteger> t1 in table1)
                {
                    if (table1[t1.Key] == table2[v])
                    {
                        return (H * t1.Key - v) % (p - 1);
                    }
                }
            }
            return -1;
        }
    }
public class PohligHellmanAlgorithm
 {
     public static BigInteger Solve(BigInteger a, BigInteger b, BigInteger
p)
     {
         //1 find dividers of (p-1)
         Dictionary<BigInteger, int> q_alpha = BigMath.Q_Alpha(p - 1);
```

```
            //2 table r
            BigInteger q;
            // get index throught the loop
            BigInteger[][] r = new BigInteger[q_alpha.Count][];
            for (int q_index = 0; q_index < q_alpha.Count; q_index++)
            {
                q = q_alpha.ElementAt(q_index).Key;
                r[q_index] = new BigInteger[(int)q]; //TODO change
                for (int j = 0; j < q; j++)
                {
                    r[q_index][j] = BigMath.Pow(a, (j * (p - 1) / q)) % p; //
r +
                }
            }
            //3 system x
            Dictionary<BigInteger, BigInteger> q_x = new
Dictionary<BigInteger, BigInteger>();
            for (int q_index = 0; q_index < q_alpha.Count; q_index++)
            {
                q = q_alpha.ElementAt(q_index).Key;
                BigInteger[] xi = new BigInteger[q_alpha[q]];
                BigInteger temp; // b^power OR b*a^power

                for (int al = 0; al < q_alpha[q]; al++)
                {
                    if (al == 0)
                    {
                        temp = BigMath.Pow(b, (p - 1) / q) % p;
                        for (int j = 0; j < q; j++)
                        {
                            if (r[q_index][j] == temp) //search in table r
                            {
                                xi[al] = j; //x0 +
                            }
                        }
                    }
                    else
```

```
                    {
                        BigInteger power = xi[0];
                        for (int i = 1; i < al; i++) {
                            power += xi[i] * (int)BigMath.Pow(q, i);
                        }

                        if (power == 1)
                        {
                            temp = b / a;
                        }
                        else
                        {
                            temp = b * BigMath.Pow(a, -power);//a^(-x0-x1..)
                        }

                        temp = BigMath.Pow(temp, (p - 1) / (BigMath.Pow(q, al
+ 1))); //(b*a)^(...)
                        temp = temp % p; //(mod p)
                        for (int j = 0; j < q; j++)
                        {
                            if (r[q_index][j] == temp) //search in table r
                            {
                                xi[al] = j; //xi +
                            }
                        }
                    }
                }
                BigInteger x = xi[0];
                for (int i = 1; i < xi.Length; i++)
                {
                    x += xi[i] * BigMath.Pow(q, i);
                }

                x = x % (BigMath.Pow(q, q_alpha[q]));
                q_x.Add(q, x); // x, q +
            }

            //4 solve system x by Chinese remainder Th
```

```
    BigInteger X = 0;
    BigInteger M0 = 1;
    BigInteger[] Mi = new BigInteger[q_x.Count];
    BigInteger[] Yi = new BigInteger[q_x.Count];
    BigInteger[] mi = new BigInteger[q_x.Count];

    int counter = 0;
    foreach (var qx in q_x)
    {
        mi[counter] = BigMath.Pow(qx.Key, q_alpha[qx.Key]);
        M0 *= mi[counter];
        counter++;
    }

    counter = 0;
    foreach (var qx in q_x)
    {
        Mi[counter] = M0 / mi[counter];
        counter++;
    }

    counter = 0;
    foreach (var qx in q_x)
    {
        for (int i = 1; i < mi[counter]; i++)
        {
            if ((Mi[counter] * i - qx.Value) % mi[counter] == 0 )
            {
                Yi[counter] = i;
                break;
            }
        }
        X += Mi[counter] * Yi[counter];
        counter++;
    }
    return X;
}
```

```
 }
class PollardRhoAlgorithm
 {
     public static BigInteger Solve(BigInteger _a, BigInteger _b,
BigInteger p)
     {
         BigInteger r = _a, q = _b;
         BigInteger x = 1, a = 0, b = 0;
         BigInteger X = x, A = a, B = b;

         for (int iterator = 1; iterator < p; iterator++)
         {
             RefreshValues(ref x, ref a, ref b, r, q, p);
             RefreshValues(ref X, ref A, ref B, r, q, p); //2i - 2
             RefreshValues(ref X, ref A, ref B, r, q, p);

             if (x == X)
             {
                 BigInteger m = (a - A).Mod(p - 1),  // + +
                     n = (B - b).Mod(p - 1);

                 if (m == 0)
                 {
                     return -1;
                 }

                 BigInteger gcd = BigMath.GCD_EuclideanExtended(m, p - 1,
out BigInteger mu, out BigInteger pu);
                 BigInteger temp =  (mu * n).Mod(p - 1);

                 for (BigInteger w = 0; w <= gcd; w++)
                 {
                     BigInteger result = ((temp + w * (p - 1)) / gcd).Mod(p
- 1);
                     if (BigMath.Pow(r, result) % p == q)
                     {
                         return result;
                     }
```

```
                    else if (w % 2 == 0 && BigMath.Pow(-r, result).Mod(p)
== q)
                    {
                        return result;
                    }
                }

                return -1;
            }
        }

        return -1;
    }

    private static void RefreshValues(ref BigInteger x, ref BigInteger a,
ref BigInteger b,
        BigInteger r, BigInteger q, BigInteger p)
    {
        if ((0 <= x) && (x <= p / 3))
        {
            x = q * x;
            a++;
            //b = b;
        }
        else if ((p / 3 < x) && (x <= 2 * p / 3))
        {
            x = x * x;
            a = 2 * a;
            b = 2 * b;
        }
        else if ((2 * p / 3 < x) && (x <= p))
        {
            x = r * x;
            //a = a;
            b++;
        }

        x = x % (p);
```

```
            a = a % (p - 1);
            b = b % (p - 1);
        }
    }
class PollardRhoAlgorithm_Modification
    {
        public static BigInteger Solve(BigInteger _a, BigInteger _b,
BigInteger p)
        {
            BigInteger r = _a, q = _b;
            BigInteger x = 1, a = 0, b = 0;
            BigInteger X = x, A = a, B = b;

            for (int iterator = 1; iterator < p; iterator++)
            {
                RefreshValues(ref x, ref a, ref b, r, q, p);
                RefreshValues(ref X, ref A, ref B, r, q, p); //2i - 2
                RefreshValues(ref X, ref A, ref B, r, q, p);

                if (x == X)
                {
                    BigInteger m = (a - A).Mod(p - 1),
                        n = (B - b).Mod(p - 1);

                    for (BigInteger i = 1; i < p; i++)
                    {
                        BigInteger temp = m * i % (p - 1);
                        if (temp == n)
                        {
                            return i;
                        }
                    }

                    return -1;
                }
            }

            return -1;
```

```
        }

        private static void RefreshValues(ref BigInteger x, ref BigInteger
a, ref BigInteger b,
            BigInteger r, BigInteger q, BigInteger p)
        {
            int i = (int)x % 3;
            switch (i)
            {
                case 0:
                    x = r * x;
                    b++;
                    break;
                case 1:
                    x = q * x;
                    a++;
                    break;
                case 2:
                    x = x * x;
                    a = 2 * a;
                    b = 2 * b;
                    break;
                default:
                    break;
            }

            x = x % (p);
            a = a % (p - 1);
            b = b % (p - 1);
        }
    }
class SimpleFormula
    {
        public static BigInteger Solve(BigInteger a, BigInteger b,
BigInteger p)
        {
            BigInteger counter = 0;
            BigInteger sum = 0;
```

```
            for (BigInteger j = 1; j <= p - 2; j++)
            {
                sum += BigMath.Pow(b, j) / (BigMath.Pow(a, j) - 1);
                counter++;
            }

            //sum <= (sum) < sum + counter, because / is about integer
division
            for (BigInteger i = 0; i <= counter; i++)
            {
                BigInteger result = (sum + i) % (p - 1);

                if (BigMath.Pow(a, result) % p == b)
                {
                    return result;
                }
            }

            return -1;
        }


    }
public static class BigMath
    {
        public static BigInteger Sqrt(BigInteger number)
        {
            BigInteger t;
            if (number < 2)
                return 1;
            BigInteger squareRoot = number / 2;
            do
            {
                t = squareRoot;
                squareRoot = (t + (number / t)) / 2;
            } while (BigInteger.Abs(t - squareRoot) > 1);

            return squareRoot;
```

```
}

public static BigInteger Pow(BigInteger number, BigInteger power)
{
    if (power < 0)
        return -1;
    if (power == 0)
        return 1;
    BigInteger res = number;
    for (BigInteger i = 1; i < power; i++)
        res *= number;

    return res;
}

//extension
public static BigInteger Mod(this BigInteger a, BigInteger p)
{
    BigInteger result = a % p;
    if (result < 0)
    {
        result += p;
    }

    return result;
}

public static int OrderOfMagnitude(BigInteger number)
{
    BigInteger num = number;
    int order = 0;
    while (num / 10 != 0)
    {
        num /= 10;
        order++;
    }
    return order;
```

```
        }

        public static BigInteger Random(int length, BigInteger maxN)
        {
            if (length <= 0)
                return -1;

            RNGCryptoServiceProvider rngCsp = new
RNGCryptoServiceProvider();
            BigInteger R;
            byte[] randBytes = maxN.ToByteArray();
            do
            {
                rngCsp.GetBytes(randBytes);
                randBytes[randBytes.Length - 1] &= 0x7F; //force sign bit
to positive
                R = new BigInteger(randBytes);
            } while (R >= maxN);
            rngCsp.Dispose();

            return BigInteger.Abs(new BigInteger(randBytes));
        }

        public static BigInteger Random(int length)
        {
            if (length <= 0)
                return -1;

            RNGCryptoServiceProvider rngCsp = new
RNGCryptoServiceProvider();
            byte[] randBytes = new byte[length];
            rngCsp.GetBytes(randBytes);
            rngCsp.Dispose();

            return BigInteger.Abs(new BigInteger(randBytes));
        }

        public static bool IsPrime(BigInteger number)
        {
```

```
            if (number == 2 || number == 3)
            {
                return true;
            }
            for (int i = 5; i < number / 2; i++)
            {
                if (number % i == 0)
                {
                    return false;
                }
            }
            return true;
        }

        //Дільники (q) числа (number) і їх кількість (alpha)
        public static Dictionary<BigInteger, int> Q_Alpha(BigInteger
number)
        {
            Dictionary<BigInteger, int> q_alpha = new
Dictionary<BigInteger, int>();
            for (int i = 2; i <= number; i++)
            {
                if (number % i == 0)
                {
                    number /= i;
                    if (q_alpha.ContainsKey(i))
                        q_alpha[i]++;
                    else
                        q_alpha.Add(i, 1);
                    i--;
                }
            }

            return q_alpha;
        }

        public static BigInteger GCD_EuclideanExtended(BigInteger n1,
BigInteger n2, out BigInteger x, out BigInteger y)
        {
```

```
        BigInteger a = n1, b = n2;

        if (b < a)
        {
            var t = a;
            a = b;
            b = t;
        }

        if (a == 0)
        {
            x = 0;
            y = 1;
            return b;
        }

        BigInteger gcd = GCD_EuclideanExtended(b % a, a, out x, out y);

        BigInteger newY = x;
        BigInteger newX = y - (b / a) * x;

        x = newX;
        y = newY;
        return gcd;
    }
}
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО"

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

МОДИФІКОВАНИЙ МЕТОД ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

Виконала: Мірошник Віталіна Ігорівна

Керівник: доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета роботи: розроблення модифікованого методу дискретного логарифмування, який буде мати кращі часові показники, ніж існуючий при програмній реалізації

Завдання:

1. Проаналізувати існуючі методи дискретного логарифмування та модифікувати один із них для зменшення часу роботи
2. Розробити програмне забезпечення для дискретного логарифмування
3. Провести тестування розробленого ПЗ
4. Провести експериментальні дослідження за допомогою розробленого ПЗ

АКТУАЛЬНІСТЬ

1. Задача знаходження дискретного логарифму в наш час вважається однією з найбільш складних та ресурсозатратних.
2. Проблема задачі дискретного логарифмування є одною із основ, на яких базується побудова криптографічних систем з відкритим ключем.
3. Пришвидшення алгоритмів знаходження дискретного логарифму має застосування у побудові більш стійких до атак криптосистем.

ПРЕДМЕТ ДОСЛІДЖЕННЯ

Процес розв'язання задачі дискретного логарифмування

ОБ'ЄКТ ДОСЛІДЖЕННЯ

Методи дискретного логарифмування

ПРИНЦИП РОБОТИ АСИМЕТРИЧНОЇ КРИПТОСИСТЕМИ

ОБГРУНТУВАНЯ ЗАСТОСУВАННЯ ОПЕРАЦІЇ MOD

Завдяки використанню mod з великими числами можна створити односторонні функції, що дозволяють легко обчислювати значення тільки в одному напрямку.

$$m^n \equiv x(\mathrm{mod}\, p) \qquad a^x \equiv b(\mathrm{mod}\, p)$$

ДЕТЕРМІНОВАНІ МЕТОДИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

Задача: $a^x \equiv b(\mathrm{mod}\ p)$

1. Метод перебору

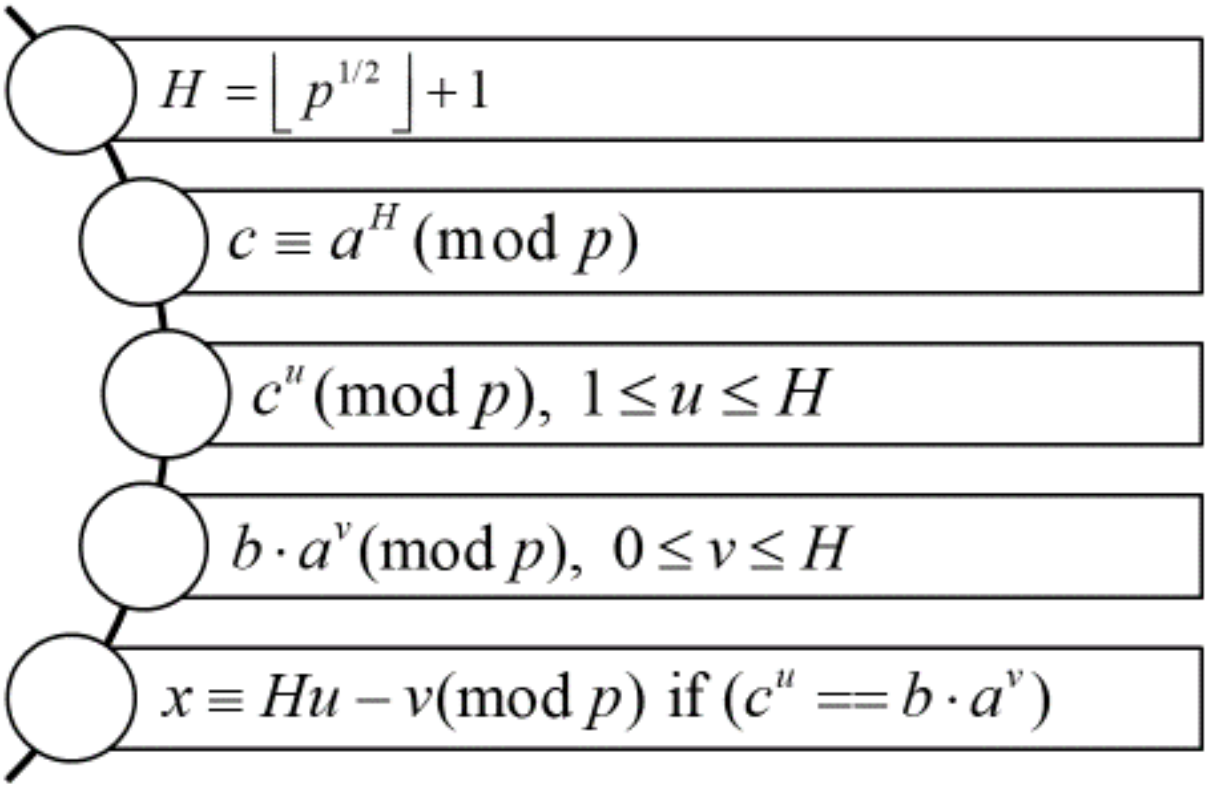
   Перебір усіх можливих рішень

2. Формула, виведена практичним шляхом

   $\log_a b \equiv \sum_{j=1}^{p-2} \frac{b^j}{1-a^j} (\mathrm{mod}\ p-1).$

ДЕТЕРМІНОВАНІ МЕТОДИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

3. Алгоритм узгодження



ДЕТЕРМІНОВАНІ МЕТОДИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

4. Алгоритм Поліга-Геллмана

$$p-1=\prod_{i=1}^{s} q_i^{\alpha_i}$$

$$r_{q,j} = a^{j(p-1)/q} \pmod p,\ j \in [0, q-1]$$

$$x_0 :\ b^{(p-1)/q} \equiv a^{x_0(p-1)/q} \pmod p$$

$$x_i :\ (ba^{-x_0 - x_1 q - \ldots - x_{i-1} q^{i-1}})^{(p-1)/q^{i+1}} \equiv a^{x_i (p-1)/q} \pmod p$$

$$x \equiv \log_a b \pmod{q^\alpha} \equiv x_0 + x_1 q + \ldots + x_{\alpha-1} q^{\alpha-1} \pmod{q^\alpha}$$

$$\log_a b \pmod{p-1} \text{ from } \log_a b \pmod{q_i^{\alpha_i}}$$

ДЕТЕРМІНОВАНІ МЕТОДИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

5. ϱ-метод Поларда

$$u_0 = v_0 = 0, \quad z_0 = 1; \quad z_{i+1} = b^{N_{i+1}} a^{V_{i+1}} \pmod{p-1};$$

$$u_{i+1} = \begin{cases} u_i + 1 \pmod{p-1}, & 0 < z_i \le p/3; \\ 2u_i \pmod{p-1}, & p/3 < z_i \le \frac{2}{3}p; \\ u_i \pmod{p-1}, & \frac{2}{3}p < z_i \le p; \end{cases} \quad v_{i+1} = \begin{cases} v_i \pmod{p-1}, & 0 < z_i \le p/3; \\ 2v_i \pmod{p-1}, & p/3 < z_i \le \frac{2}{3}p; \\ v_i + 1 \pmod{p-1}, & \frac{2}{3}p < z_i \le p; \end{cases}$$

$$z_i = z_{2i}, m = u_i - u_{2i}, n = v_{2i} - v_i \text{ from } (z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$$

$$d = \gcd(m, p-1) = \lambda \times m + \mu \times (p-1)$$

$$b \equiv a^k \theta^w \pmod{p}, \ \theta \equiv a^{(p-1)/d}, \ w \in [0; d]$$

$$x \equiv (k + w \times (p-1)) / d$$

ДЕТЕРМІНОВАНІ МЕТОДИ ДИСКРЕТНОГО ЛОГАРИФМУВАННЯ

5. ϱ-метод Поларда – запропонована модифікація

$$u_0 = v_0 = 0, \quad z_0 = 1; \quad \alpha = z_i \pmod 3;$$

$$u_{i+1} = \begin{cases} u_i \pmod{p-1}, & \alpha = 0; \\ u_i + 1 \pmod{p-1}, & \alpha = 1; \\ 2u_i \pmod{p-1}, & \alpha = 2; \end{cases} \quad v_{i+1} = \begin{cases} v_i + 1 \pmod{p-1}, & \alpha = 0; \\ v_i \pmod{p-1}, & \alpha = 1; \\ 2v_i \pmod{p-1}, & \alpha = 2; \end{cases}$$

$$z_{i+1} = \begin{cases} z_i \times a \pmod p, & \alpha = 0; \\ z_i \times b \pmod p, & \alpha = 1; \\ z_i^2 \pmod p, & \alpha = 2; \end{cases}$$

$$z_i = z_{2i}, m = u_i - u_{2i}, n = v_{2i} - v_i \text{ from } (z_i, u_i, v_i, z_{2i}, u_{2i}, v_{2i})$$

$$a^m \equiv b^n \pmod p$$

$$m \times x \equiv n \pmod{p-1}, \; x \in [1; p)$$

ЗАСОБИ РЕАЛІЗАЦІЇ

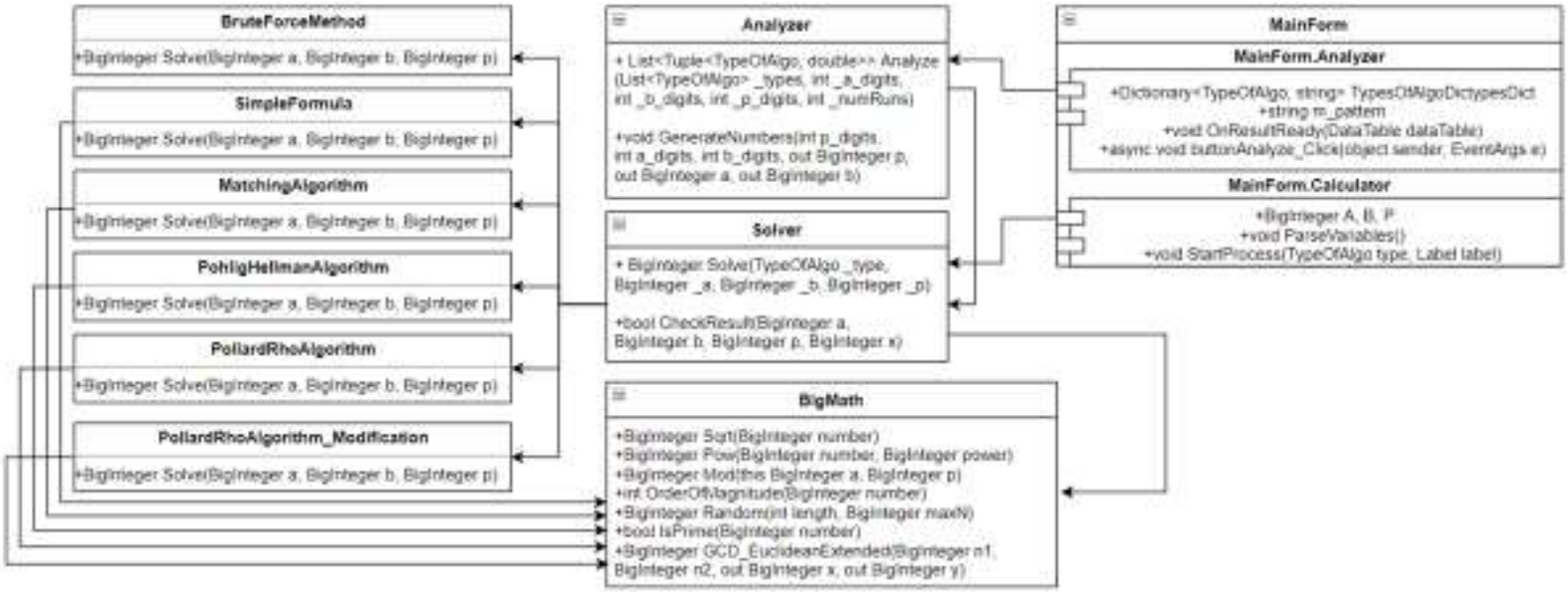
Мова програмування: C#

Фреймворк: .NET Framework

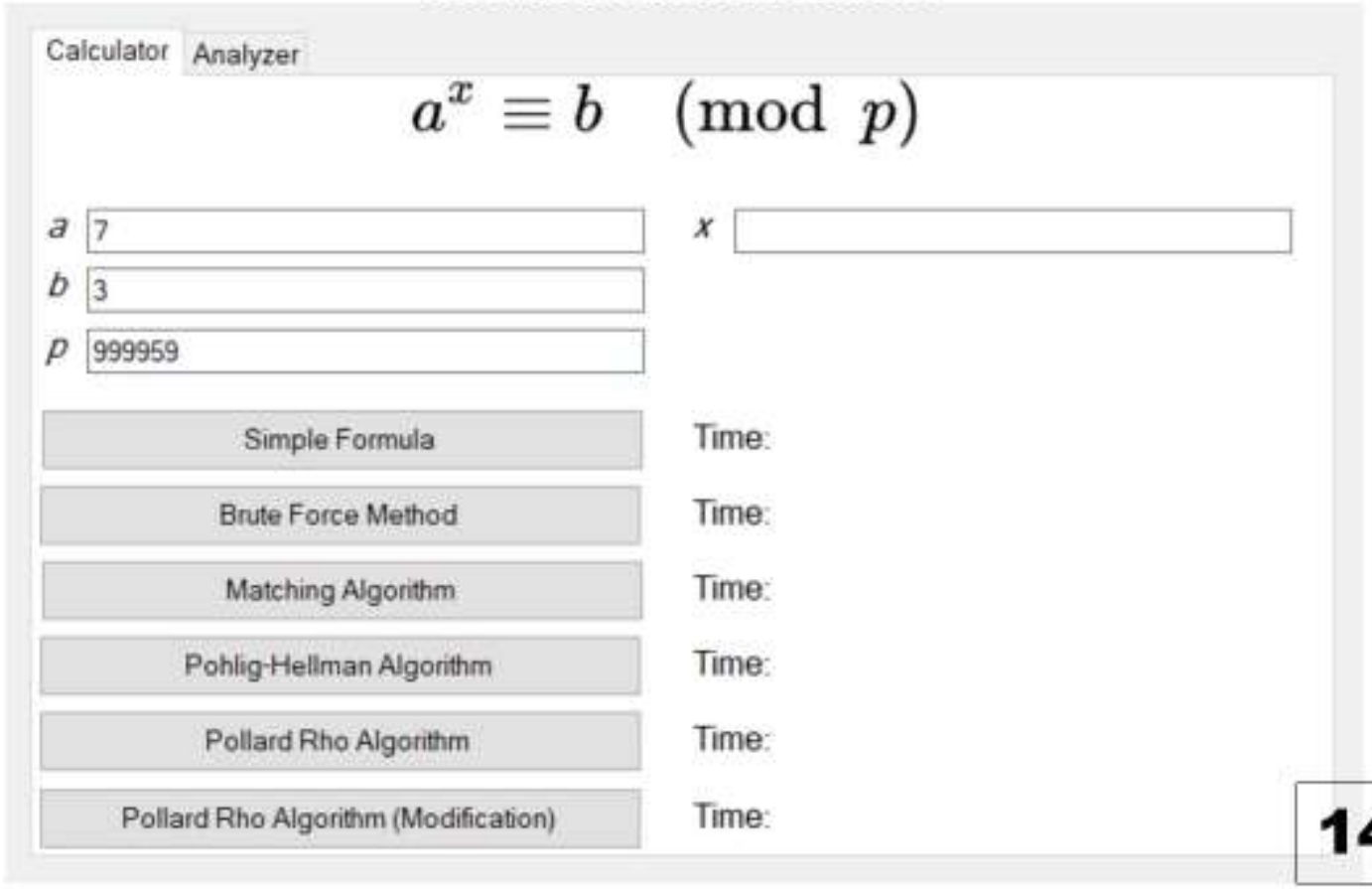
API: Windows Forms

IDE: Microsoft Visual Studio 2019

ДІАГРАМА КЛАСІВ

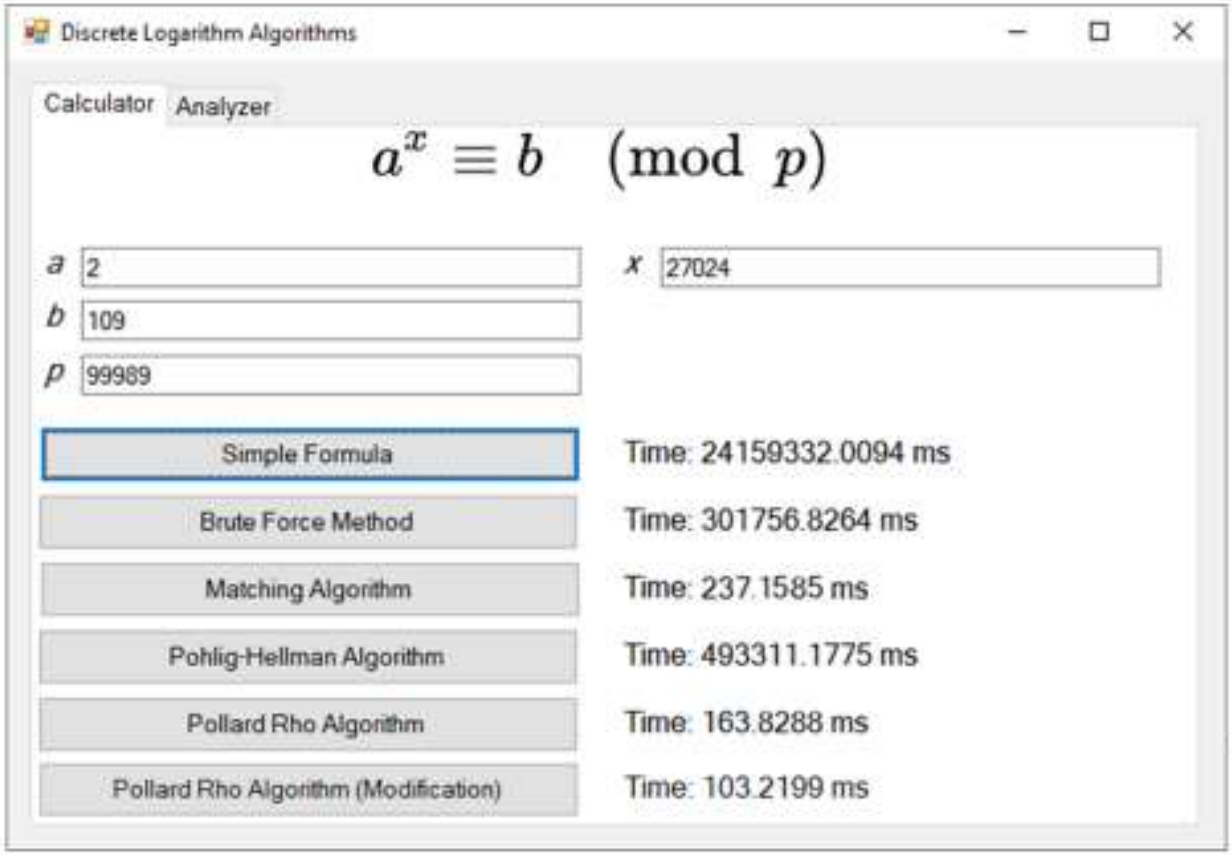


ДЕМОНСТРАЦІЯ РОБОТИ КАЛЬКУЛЯТОРА

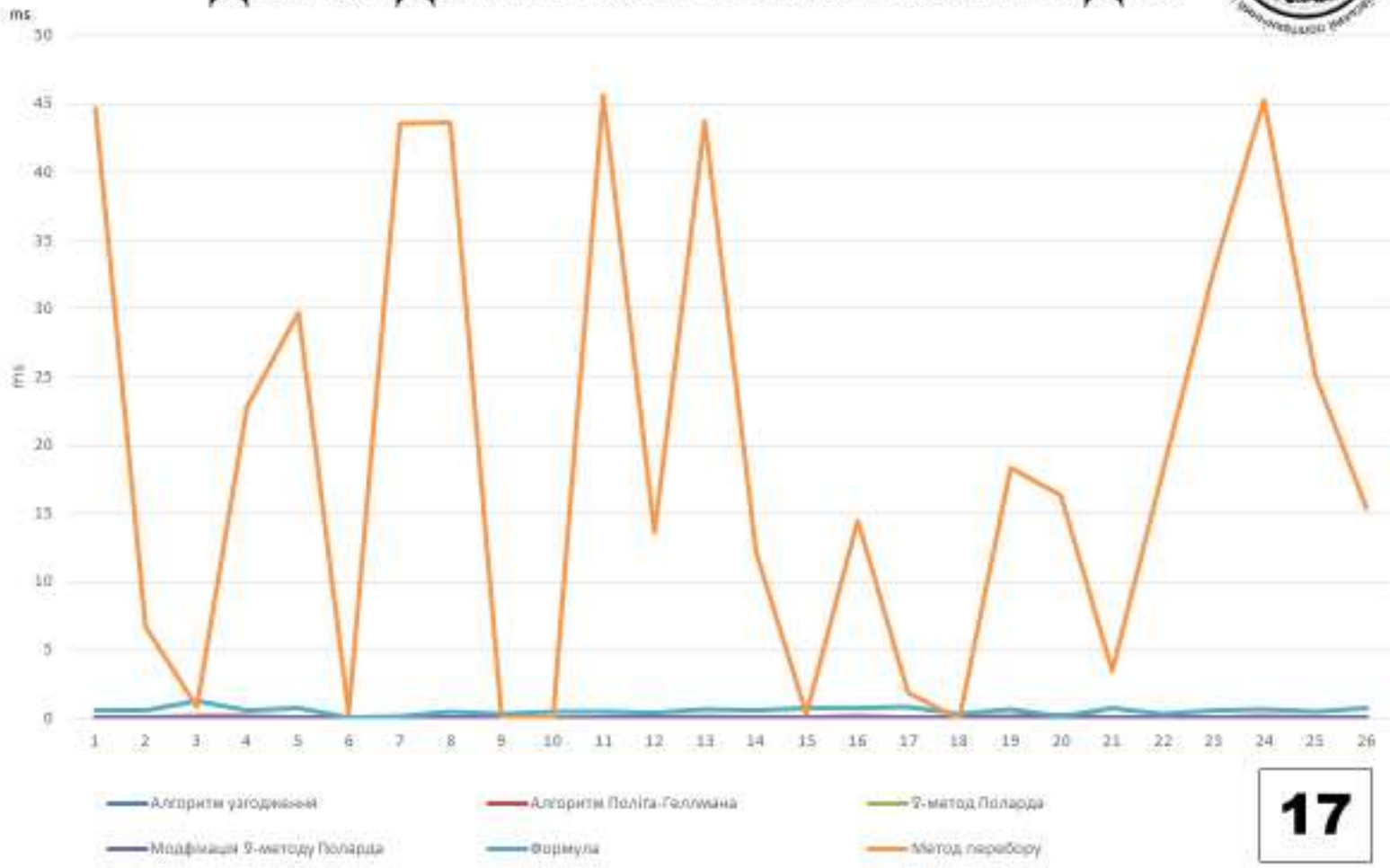


ДЕМОНСТРАЦІЯ РОБОТИ АНАЛІЗАТОРА

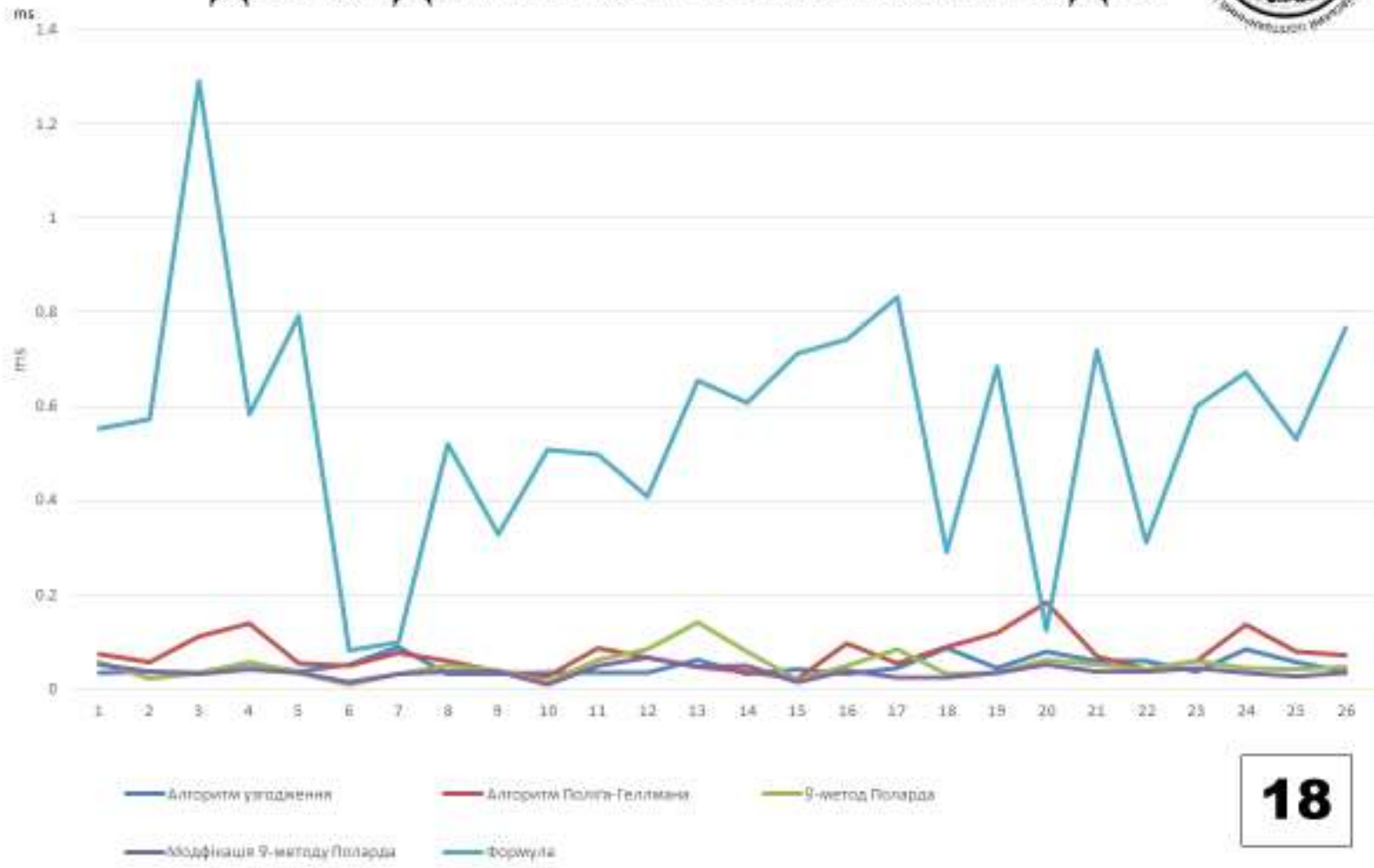
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОБОТИ МЕТОДІВ



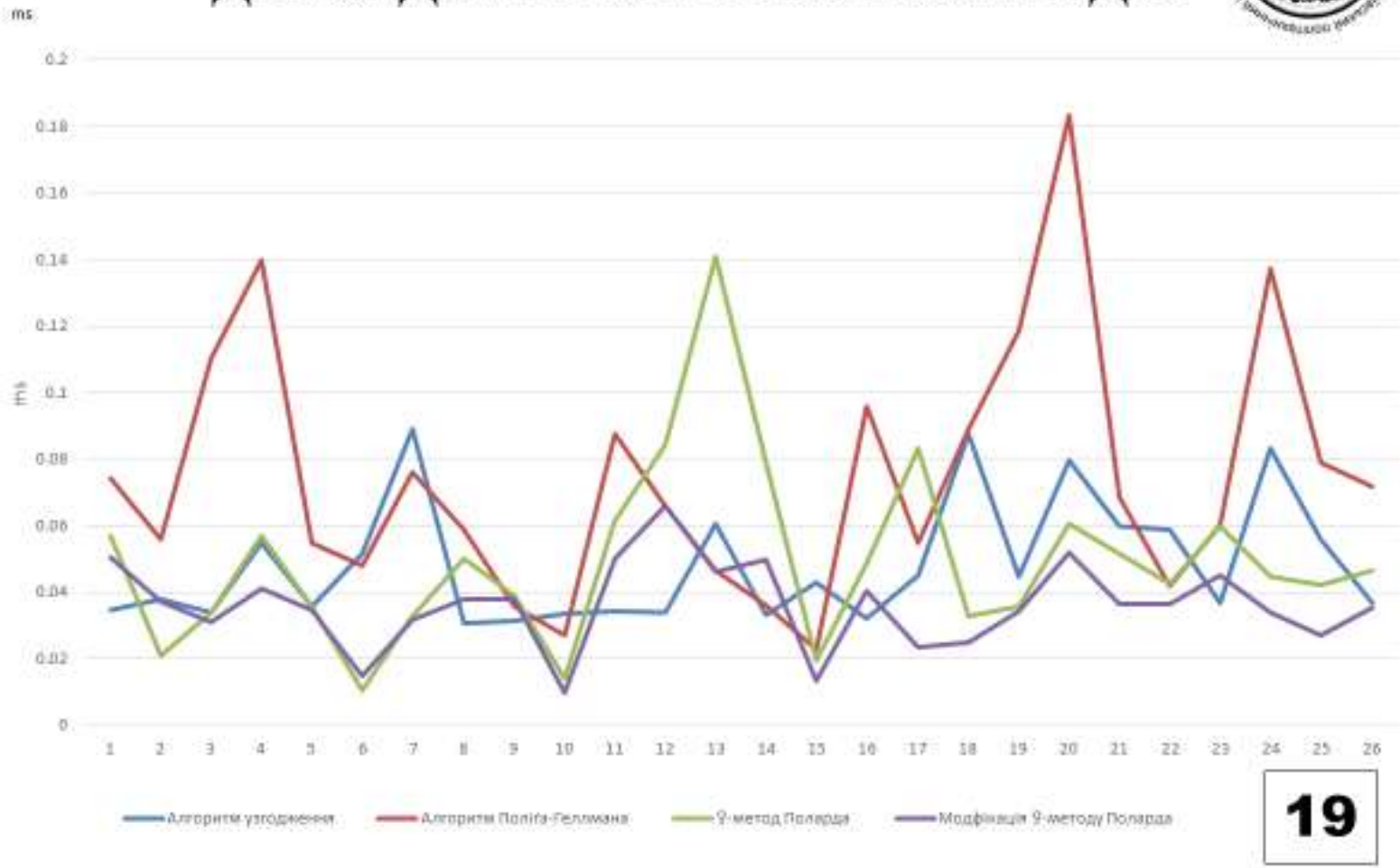
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОБОТИ МЕТОДІВ



ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОБОТИ МЕТОДІВ



ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОБОТИ МЕТОДІВ



ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ КЛАСИЧНОГО ТА МОДИФІКОВАНОГО ϱ-МЕТОДУ ПОЛАРДА

НАУКОВА НОВИЗНА

Запропоновано модифікований ϱ-метод Поларда, який відрізняється від існуючого тим, що при розбитті на псевдовипадкові множини використовується модулярна арифметика замість арифметики з плаваючою крапкою, а також відсутністю операції піднесення до степеня на останньому кроці, що забезпечує зменшення часу роботи в 1.5 рази порівняно з існуючим методом для простого модуля довжиною 2 байти.

ВИСНОВКИ

1. Проаналізовано відомі методи дискретного логарифмування
2. Запропоновано модифікацію ϱ-метода Поларда
3. Розроблено програмне забезпечення для обчислення та аналізу методів дискретного логарифмування
4. Протестовано розроблене програмне забезпечення та проведені експериментальні дослідження часу роботи алгоритмів

Дякую за увагу!